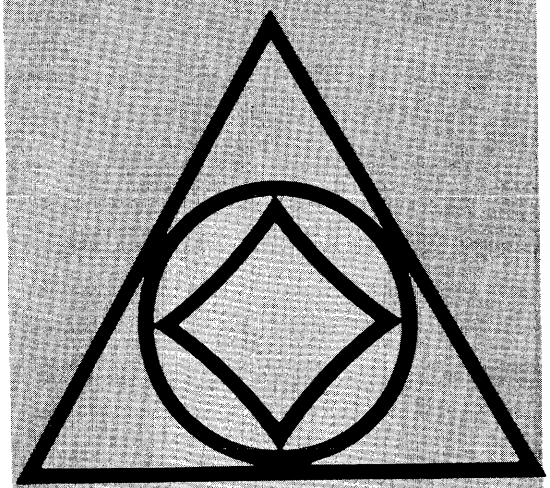


AFIPS

CONFERENCE PROCEEDINGS

VOLUME 25

1964

SPRING JOINT COMPUTER CONFERENCE

1964
SPARTAN BOOKS, INC.
Baltimore, Md.
CLEAVER-HUME PRESS
London

The ideas and opinions expressed herein are solely those of the authors and are not necessarily representative of or endorsed by the 1964 Spring Joint Computer Conference Committee or the American Federation of Information Processing Societies.

Library of Congress Catalog Card Number: 55–44701

Sole Distributors in Great Britain, the British Commonwealth and the Continent of Europe:

CLEAVER-HUME PRESS
10-15 St. Martins Street
London W. C. 2

LIST OF JOINT COMPUTER CONFERENCES

1. 1951 Joint AIEE-IRE Computer Conference, Philadelphia, December 1951
2. 1952 Joint AIEE-IRE-ACM Computer Conference, New York, December 1952
3. 1953 Western Computer Conference, Los Angeles, February 1953
4. 1953 Eastern Joint Computer Conference, Washington, December 1953
5. 1954 Western Computer Conference, Los Angeles, February 1954
6. 1954 Eastern Joint Computer Conference, Philadelphia, December 1954
7. 1955 Western Joint Computer Conference, Los Angeles, March 1955
8. 1955 Eastern Joint Computer Conference, Boston, November 1955
9. 1956 Western Joint Computer Conference, San Francisco, February 1956
10. 1956 Eastern Joint Computer Conference, New York, December 1956
11. 1957 Western Joint Computer Conference, Los Angeles, February 1957
12. 1957 Eastern Joint Computer Conference, Washington, December 1957
13. 1958 Western Joint Computer Conference, Los Angeles, May 1958
14. 1958 Eastern Joint Computer Conference, Philadelphia, December 1958
15. 1959 Western Joint Computer Conference, San Francisco, March 1959
16. 1959 Eastern Joint Computer Conference, Boston, December 1959
17. 1960 Western Joint Computer Conference, San Francisco, May 1960
18. 1960 Eastern Joint Computer Conference, New York, December 1960
19. 1961 Western Joint Computer Conference, Los Angeles, May 1961
20. 1961 Eastern Joint Computer Conference. Washington, December 1961
21. 1962 Spring Joint Computer Conference, San Francisco, May 1962
22. 1962 Fall Joint Computer Conference, Philadelphia, December 1962
23. 1963 Spring Joint Computer Conference, Detroit, May 1963
24. 1963 Fall Joint Computer Conference, Las Vegas, November 1963
25. 1964 Spring Joint Computer Conference, Washington, April 1964

Conferences 1 to 19 were sponsored by the National Joint Computer Committee, predecessor of AFIPS. Back copies of the proceedings of these conferences may be obtained, if available, from:

- Association for Computing Machinery, 14 E. 69th St., New York 21, N. Y.
- American Institute of Electrical Engineers, 345 E. 47th St., New York 17, N. Y.
- Institute of Radio Engineers, 1 E. 79th St., New York 21, N. Y.

Conferences 20 and up are sponsored by AFIPS. Copies of AFIPS Conference Proceedings may be ordered from the publishers as available at the prices indicated below. Members of societies affiliated with AFIPS may obtain copies at the special "Member Price" shown.

Volume	*List Price*	*Member Price*	*Publisher*
20	$12.00	$7.00	Macmillan Co., 60 Fifth Ave., New York 11, N. Y.
21	6.00	6.00	National Press, 850 Hansen Way, Palo Alto, Calif.
22	8.00	4.00	Sparatan Books, Inc., 301 N. Charles St., Baltimore 1, Md.
23	10 00	5.00	Sparatan Books, Inc.
24	16.50	8.25	Sparatan Books, Inc.
25			Sparatan Books, Inc.

NOTICE TO LIBRARIANS

This volume (25) continues the Joint Computer Conference Proceedings (LC55–44701) as indicated in the above table. It is suggested that the series be filed under AFIPS and cross referenced as necessary to the Eastern, Western, Spring, and Fall Joint Computer Conferences.

CONTENTS

PREFACE

The following pages contain the papers which were presented and discussed in the formal technical and tutorial sessions of the 1964 Spring Computer Conference. The Conference theme, "Computers '64: Problem-Solving in a Changing World," is intended to suggest the significance of this year as the mid-point between the infancy of electronic digital computation (the first elements of ENIAC began operating in June 1944) and the Orwellian year 1984, which symbolizes to some critics an unhappy potential which might be achieved if we do not wisely guide our rapidly advancing technology. Society is increasingly concerned with the broad adjustments that must take place if man is to gain the maximum long-term advantage from the computer. Reflections of this concern and of the increasing uses to which computers are being applied are, among others, the papers dealing with social implications and the tutorial papers on hybrid systems, compilers, and information retrieval.

This volume, a product of the 25th in the series of Joint Computer Conferences, for the first time includes an index. Thanks for this useful addition are due to the authors and session chairmen, who were most cooperative in getting "final copy" submitted on schedule, and to our publisher, who assumed the burden of preparing the index. Appreciation must also be expressed for the contributions of many other persons, who participated as conference planners, panel members, and reviewers. A special acknowledgement is made to the members of the Technical Program Committee, who willingly assumed the heavy burden of assembling the formal program.

HERBERT R. KOLLER
General Chairman
1964 Spring Joint Computer Conference

PROGRAMMING SYSTEMS AND LANGUAGES
A Historical Survey

Saul Rosen
Professor of Mathematics and Computer Sciences
Purdue University
West Lafayette, Indiana

1. *Introduction.* Twenty years ago, in 1943, there were no Electronic computers. Ten years ago, in 1953, a large number of Electronic calculators were in use, but the general purpose stored program electronic computer was still quite rare. The Coincident Current Magnetic Core memory which finally provided both reliability and speed at reasonable cost had only just been developed, and was still a laboratory device. A number of specially designed, mostly one of a kind, computers were in operation at Universities and government research centers. Commercially, a few Univac I computers had been delivered and were operating with great reliability at rather low speed. A few IBM 701's provided high speed but with very poor reliability. In 1953 most computing was being done by the Card-Programmed Calculator, an ingenious mating of an Electromechanical Accounting Machine with an Electronic Calculating Punch.

Between 1954 and 1958 many hundreds of Electronic computers were installed. This was the era of the Vacuum Tube Computer, with Magnetic Drum storage on lower priced machines, and Magnetic Core storage on the larger more expensive ones. By 1959 the first transistorized computers had been delivered, and the production of vacuum tube computers ceased almost immediately. Low cost magnetic core memories made the magnetic drum almost obsolete except as auxiliary storage. Since 1959 thousands of computers have been delivered and installed, including many hundreds of very large computers. Electronic Computing and Data-processing have become part of the everyday industrial and commercial environment, and Electronic Computer manufacturing and programming has become a multi-billion dollar industry.

Because of this rapid, almost explosive pattern of growth most systems, both in the hardware and software area could not be adequately planned, and it was often not possible to make adequate evaluations of old systems in time to use such evaluations in the design of new ones.

2. *Developments up to 1957.* The first programming systems were systems of subroutines. Subroutines were in use on the pre-electronic Mark I, a large electromechanical computer built as a joint effort by IBM and Harvard University in the early forties.[1] The EDSAC at the University of Manchester was probably the first stored program Electronic Computer in operation (1949). The classic text on programming the EDSAC, by Wilkes, Wheeler and Gill[2] makes the subroutine library the basis of programming, and stresses the automatic relocation feature of the EDSAC, which makes such libraries easy to use.

The IBM Card-Programmed Calculator, developed between 1947 and 1949, was not very fast by modern standards, but was an extremely versatile device. Operation codes were deter-

mined by the wiring of control boards. Ingenuity could be expended on designing boards to optimize performance on particular problems that taxed the capacity of the computer, or it could be expended on the design of general purpose boards for use in large classes of problems. A set of general purpose boards represented a language for the CPC, and a number of languages were tried and used.[3] Most Scientific installations finally settled on a wiring that made the CPC appear to be a floating point machine with three-address logic, and with a standard vocabulary of built-in routines like Square Root, Sine, Exponential, etc. Experience with the CPC systems had many influences on the programming systems that were designed for the stored-program computers that followed.

One of the most important of the early automatic programming groups was associated with the Whirlwind project at MIT. Whirlwind, which was built between 1947 and 1951, was a fast but very basic computer. With only a 16 bit word it had a very limited instruction code, and very limited high speed storage. Even relatively simple calculations required the use of multi-precision techniques. The very difficulty of using the machine in its own language provided great incentive toward the development of programming languages. The Summer Session Computer at MIT was one of the early interpretive systems, designed to make the Whirlwind computer available to students at a summer course in computing at MIT. These early developments led to the design of a quite elaborate "Comprehensive System" for Whirlwind.[4] At the associated Lincoln Laboratories the groundwork was laid for the very large programming systems that were developed in connection with Sage and other military projects.[5]

The first large scale electronic computer available commercially was the Univac I (1951. The first Automatic Programming group associated with a commercial computer effort was the group set up by Dr. Grace Hopper in what was then the Eckert-Mauchly Computer Corp., and which later became the Univac Division of Sperry Rand. The Univac had been designed so as to be relatively easy to program in its own code. It was a decimal, alphanumeric machine, with mnemonic instructions that were easy to remember and use. The 12 character word made scaling of many fixed-point calculations fairly easy. It was not always easy to see the advantage of assembly systems and compilers that were often slow and clumsy on a machine with only 12,000 characters of high speed storage (200 microseconds average access time per 12 character word). In spite of occasional setbacks, Dr. Hopper persevered in her belief that programming should and would be done in problem-oriented languages. Her group embarked on the development of a whole series of languages, of which the most used was probably A2, a compiler that provided a three address floating point system by compiling calls on floating point subroutines stored in main memory.[6,7] The Algebraic Translator AT3 (Math-Matic[8]) contributed a number of ideas to Algol and other compiler efforts, but its own usefulness was very much limited by the fact that Univac had become obsolete as a scientific computer before AT3 was finished. The BO (Flow-Matic[8,9]) compiler was one of the major influences on the COBOL language development which will be discussed at greater length later. The first sort generators[10] were produced by the Univac programming group in 1951. They also produced what was probably the first large scale symbol manipulation program, a program that performed differentiation of formulas submitted to it in symbolic form.[11]

Another and quite independent group at Univac concerned itself with an area that would now be called computer-oriented compilers. Anatol Holt and William Turanski developed a compiler and a concept that they called GP (Generalized Programming[12]). Their system assumed the existence of a very general subroutine library. All programs would be written as if they were to be library programs, and the library and system would grow together. A program was assembled at compile time by the selection and modification of particular library programs and parts of library programs. The program as written by the programmer would provide parameters and specifications according to which the very general library programs would be made specific to a particular problem. Subroutines in the library were organized in hierarchies, in which subroutines at one level

could call on others at the next level. Specifications and parameters could be passed from one level to the next.

The system was extended and elaborated in the GPX system that they developed for Univac II. They were one of the early groups to give serious attention to some difficult problems relative to the structure of programs, in particular the problem of segmenting of programs, and the related problem of storage allocation.

Perhaps the most important contribution of this group was the emphasis that they placed on the programming system rather than on the programming language. In their terms, the machine that the programmer uses is not the hardware machine, but rather an extended machine consisting of the hardware enhanced by a programming system that performs all kinds of service and library maintenance functions in addition to the translation and running of programs.

The IBM 701 (1952) was the first commercially marketed large scale binary computer. The best known programming language on the 701 was Speedcode,[13,14] a language that made the one address, binary, fixed point 701 appear to be a three address decimal floating point machine with index registers. More than almost any other language, Speedcode demonstrated the extent to which users were willing to sacrifice computer speed for the sake of programming convenience.

The PACT[15,16] system on the 701 set a precedent as the first programming system designed by a committee of computer users. It also set a precedent for a situation which unfortunately has been quite common in the computer field. The computer for which the programming system was developed was already obsolete before the programming system itself was completed. PACT ideas had a great deal of influence on the later developments that took place under the auspices of the SHARE organization.

Delivery of the smaller, medium scale magnetic-drum computers started in 1953, and by 1955-56 they were a very important factor in the computer field. The IBM 650 was by far the most popular of the early drum computers. The 650 was quite easy to program in its own language, and was programmed that way in many applications, especially in the data-processing area. As a scientific computer it lacked floating point hardware, a feature that was later made available. A number of interpretive floating point systems were developed, of which the most popular was the one designed at the Bell Telephone Laboratories.[17] This was a three address floating point system with automatic looping and with built in Mathematical subroutines. It was a logical continuation of the line of systems that had started with the general purpose CPC boards, and had been continued in 701 Speedcode. It proved that on the right kind of computer an interpretive system can provide an efficient effective tool. Interpretive systems fell into disrepute for a number of years. They are making a very strong comeback at the present time in connection with a number of so-called micro-programmed computers that have recently appeared on the market.

The 650 permitted optimization of programs by means of proper placement of instructions on the drum. Optimization was a very tedious job for the programmer, but could produce a very considerable improvement in program running time. A program called SOAP, a Symbolic Optimizer and Assembly Programs, combined the features of symbolic assembly and automatic optimization. There is some doubt as to whether a symbolic assembly system would have received very general acceptance on the 650 at the time SOAP was introduced. The optimization feature was obviously valuable. Symbolic assembly by itself on a decimal machine without magnetic tape did not present advantages that were nearly as obvious.

The major competitor to the 650 among the early magnetic drum computers was the Datatron 205, which eventually became a Burroughs product. It featured a 4000 word magnetic drum storage plus a small 80 word fast access memory in high speed loops on the drum. Efficient programs had to make very frequent use of block transfer instructions, moving both data and programs to high speed storage. A number of interpretive and assembly system were built to provide programmed floating point instructions and some measure of automatic use of the

high speed loops. The eventual introduction of floating point hardware removed one of the principal advantages of most of these systems. The Datatron was the first commercial system to provide an index register and automatic relocation of subroutines, features provided by programming systems on other computers. For these reasons among others the use of machine code programming persisted through most of the productive lifetime of the Datatron 205.

One of the first Datatron computers was installed at Purdue University. One of the first Algebraic compilers was designed for the Datatron by a group at Purdue headed by Dr. Al Perlis. This is another example of a compiler effort based on a firm belief that programming should be done in problem-oriented languages, even if the computer immediately available may not lend itself too well to the use of such languages. A big problem in the early Datatron systems was the complete lack of alphanumeric input. The computer would recognize pairs of numbers as representing characters for printing on the flexowriter, but there was no way to produce the same pair of numbers by a single key stroke on any input preparation device. Until the much later development of new input-output devices, the input to the Purdue compiler was prepared by manually transcribing the source language into pairs of numbers.

When Dr. Perlis moved to Carnegie Tech the some compiler was written for the 650, and was named IT.[18] IT made use of the alphanumeric card input of the 650, and translated from a simplified algebraic language into SOAP language as output. IT and languages derived from it became quite popular on the 650, and on other computers, and have had great influence on the later development of programming languages. A language Fortransit provided translation from a subset of Fortran into IT, whence a program would be translated into SOAP, and after two or more passes through SOAP it would finally emerge as a machine language program. The language would probably have been more popular if its translation were not such an involved and time-consuming process. Eventually other, more direct translators were built that avoided many of the intermediate passes.

The 701 used a rather unreliable electrostatic tube storage system. When Magnetic core storage became available there was some talk about a 701M computer that would be an advanced 701 with core storage. The idea of a 701M was soon dropped in favor of a completely new computer, the 704. The 704 was going to incorporate into hardware many of the features for which programming systems had been developed in the past. Automatic floating point hardware and index registers would make interpretive systems like Speedcode unnecessary.

Along with the development of the 704 hardware IBM set up a project headed by John Backus to develop a suitable compiler for the new computer. After the expenditure of about 25 man years of effort they produced the first Fortran compiler.[19,20] Fortran is in many ways the most important and most impressive development in the early history of automatic programming.

Like most of the early hardware and software systems, Fortran was late in delivery, and didn't really work when it was delivered. At first people thought it would never be done. Then when it was in field test, with many bugs, and with some of the most important parts unfinished, many thought it would never work. It gradually got to the point where a program in Fortran had a reasonable expectancy of compiling all the way through and maybe even of running. This gradual change of status from an experiment to a working system was true of most compilers. It is stressed here in the case of Fortran only because Fortran is now almost taken for granted, as if it were built into the computer hardware.

In the early days of automatic programming, the most important criterion on which a compiler was judged was the efficiency of the object code. "You only compile once, you run the object program many times," was a statement often quoted to justify a compiler design philosophy that permitted the compiler to take as long as necessary, within reason, to produce good object code. The Fortran compiler on the 704 applied a number of difficult and ingenious techniques in an attempt to produce object coding that would be as good as that produced by a good programmer programming in machine

code. For many types of programs the coding produced is very good. Of course there are some for which it is not so good. In order to make effective use of index registers a very complicated index register assignment algorithm was used that involved a complete analysis of the flow of the program and a simulation of the running of the program using information obtained from frequency statements and from the flow analysis. This was very time consuming, especially on the relatively small initial 704 configuration. Part of the index register optimization fell into disuse quite early but much of it was carried along into Fortran II and is still in use on the 704/9/90. In many programs it still contributes to the production of better code than can be achieved on the new Fortran IV compiler.

Experience led to a gradual change of philosophy with respect to compilers. During debugging, compiling is done over and over again. One of the major reasons for using a problem oriented language is to make it easy to modify programs frequently on the basis of experience gained in running the programs. In many cases the total compile time used by a project is much greater than the total time spent running object codes. More recent compilers on many computers have emphasized compiling time rather than run time efficiency. Some may have gone too far in that direction.

It was the development of Fortran II that made it possible to use Fortran for large problems without using excessive compiling time. Fortran II permitted a program to be broken down into subprograms which could be tested and debugged separately. With Fortran II in full operation, the use of Fortran spread very rapidly. Many 704 installations started to use nothing but Fortran. A revolution was taking place in the scientific computing field, but many of the spokesmen for the computer field were unaware of it. A number of major projects that were at crucial points in their development in 1957-1959 might have proceeded quite differently if there was more general awareness of the extent to which the use of Fortran had been accepted in many major 704 installations. Among these are the Algol project and the SOS project which are discussed below.

3. *Algol and its Dialects.* Until quite recently, large scale computers have been mainly an American phenomenon. Smaller computers were almost worldwide right from the beginning. An active computer organization GAMM had been set up in Europe, and in 1957 a number of members of this organization were actively interested in the design of Algebraic compilers for a number of machines. They decided to try to reach agreement on a common language for various machines, and made considerable progress toward the design of such a language. There are many obvious advantages to having generally accepted computer independent problem oriented languages. It was clear that a really international effort in this direction could only be achieved with United States participation. The President of GAMM wrote a letter to John Carr who was then President of the ACM, suggesting that representatives of ACM and of GAMM meet together for the purpose of specifying an international language for the description of computing procedures.

The ACM up to that time had served as a forum for the presentation of ideas in all aspects of the computer field. It had never engaged in actual design of languages or systems.

In response to the letter from GAMM, Dr. Carr appointed Dr. Perlis as chairman of a committee on programming languages. The committee set out to specify an Algebraic compiler language that would represent the American proposal at a meeting with representatives of GAMM at which an attempt would be made to reach agreement on an internationally accepted language. The ACM committee consisted of representatives of the major computer manufacturers, and representatives of several Universities and research agencies that had done work in the compiler field. Probably the most active member of the committee was John Backus of IBM. He was probably the only member of the committee whose position permitted him to spend full time on the language design project, and a good part of the "American Proposal" was based on his work.

The ACM committee had a number of meetings without any very great sense of urgency. Subcommittees worked on various parts of the

language and reported back to the full committee, and in general there was little argument or disagreement. There is after all very general agreement about the really basic elements of an Algebraic language. Much of the language is determined by the desirability of remaining as close as possible to Mathematical notation. This is tempered by experience in the use of computers and in the design of compilers which indicates some compromises between the demands of desirable notation and those of practical implementation.

At one meeting of the committee Dr. Bauer, one of the leaders of the GAMM effort, presented a report on the proposed European language. Among other things they proposed that English language key words, like begin, end, for, do, be used as a world-wide standard. Of course this is something the American committee would never have proposed, but it seemed quite reasonable to go along with the Europeans in this matter. Although some of the notations seemed strange, there were very few basic disagreements between what GAMM was proposing, and what the ACM committee was developing. Dr. Bauer remarked that the GAMM organization felt somewhat like the Russians who were meeting with constant rebuffs in an effort to set up a summit meeting. With such wide areas of agreement why couldn't the ACM-GAMM meeting take place?

Although there is quite general agreement about the basic elements of an Algebraic language, there is quite considerable disagreement about how far such a language should go, and about how some of the more advanced and more difficult concepts should be specified in the language. Manipulation of strings of symbols, direct handling of vectors, matrices, and multiple precision quantities, ways to specify segmentation of problems, and the allocation and sharing of storage; these were some of the topics which could lead to long and lively discussion. The ACM language committee decided that it was unreasonable to expect to reach an agreement on an international language embodying features of this kind at that time. It was decided to set up two subcommittees. One would deal with the specification of a language which included those features on which it was reasonable to expect a wide range of agreement. The other was to work toward the future, toward the specification of a language that would really represent the most advanced thinking in the computer field.

The short-range committee was to set up a meeting in Europe with representatives of GAMM. Volunteers for work on this committee would have to arrange for the trip to Europe and back, and were therefore limited to those who worked for an organization that would be willing to sponsor such a trip. The ACM was asked to underwrite the trip for Dr. Perlis.

The meeting of the ACM and GAMM subcommittees was held in Zurich in the spring of 1958, and the result was a Preliminary report on an International Algebraic Language, which has since become popularly known as Algol 58.[21]

With the use of Fortran already well established in 1958, one may wonder why the American committee did not recommend that the international language be an extension of, or at least in some sense compatible with Fortran. There were a number of reasons. The most obvious has to do with the nature and the limitations of the Fortran language itself. A few features of the Fortran language are clumsy because of the very limited experience with compiler languages that existed when Fortran was designed. Most of Fortran's most serious limitations occur because Fortran was not designed to provide a completely computer independent language; it was designed as a compiler language for the 704. The handling of a number of statement types, in particular the Do and If statements, reflects the hardware constraints of the 704, and the design philosophy which kept these statements simple and therefore restricted in order to simplify optimization of object coding.

Another and perhaps more important reason for the fact that the ACM committee almost ignored the existence of Fortran has to do with the predominant position of IBM in the large scale computer field in 1957-1958 when the Algol development started. Much more so than now there were no serious competitors. In the data processing field the Univac II was much too late to give any serious competition to the IBM 705. RCA's Bizmac never really had a chance, and Honeywell's Datamatic 1000, with

its 3 inch wide tapes, had only very few specialized customers. In the Scientific field there were those who felt that the Univac 1103/1103a/1105 series was as good or better than the IBM 701/704/709. Univac's record of late delivery and poor service and support seemed calculated to discourage sales to the extent that the 704 had the field almost completely to itself. The first Algebraic compiler produced by the manufacturer for the Univac Scientific computer, the 1103a, was Unicode, a compiler with many interesting features, that was not completed until after 1960, for computers that were already obsolete. There were no other large scale scientific computers. There was a feeling on the part of a number of persons highly placed in the ACM that Fortran represented part of the IBM empire, and that any enhancement of the status of Fortran by accepting it as the basis of an international standard would also enhance IBM's monopoly in the large scale scientific computer field.

The year 1958 in which the first Algol report was published, also marked the emergence of large scale high speed transistorized computers, competitive in price and superior in performance to the vacuum tube computers in general use. At the time I was in charge of Programming systems for the new model 2000 computers that Philco was preparing to market. An Algebraic compiler was an absolute necessity, and there was never really any serious doubt that the language had to be Fortran.[22,22A] The very first sales contracts for the 2000 specified that the computer had to be equipped with a compiler that would accept 704 Fortran source decks essentially without change. Other manufacturers, Honeywell, Control Data, Bendix, faced with the same problems, came to the same conclusion. Without any formal recognition, in spite of the attitude of the professional committees, Fortran became the standard scientific computing language. Incidentally, the emergence of Fortran as a standard helped rather than hindered the development of a competitive situation in the scientific computer field.

To go on with the Algol development, the years 1958-1959 were years in which many new computers were introduced. The time was ripe for experimentation in new languages. As mentioned earlier there are many elements in common in all Algebraic languages, and everyone who introduced a new language in those years called it Algol, or a dialect of Algol. The initial result of this first attempt at the standardization of Algebraic languages was the proliferation of such languages in great variety.

A very bright young programmer at Burroughs had some ideas about writing a very fast one pass compiler for Burroughs new 220 computer. The compiler has come to be known as Balgol.

A compiler called ALGO was written for the Bendix G15 computer. At Systems Development Corporation, programming systems had to be developed for a large command and control system based on the IBM military computer (ANFSQ32). The resulting Algebraic language with fairly elaborate data description facilities was JOVIAL[23] (Jules Schwartz' own Version of the International Algebraic Language). By now compilers for JOVIAL have been written for the IBM 7090, the Control Data 1604, the Philco 2000, the Burroughs D825, and for several versions of IBM military computers.

The Naval Electronics Laboratory at San Diego was getting a new Sperry Rand Computer, the Countess. With a variety of other computers installed and expected they stressed the description of a compiler in its own language to make it easy, among other things, to produce a compiler on one computer using a compiler on another. They also stressed very fast compiling times, at the expense of object code running times, if necessary. The language was called Neliac,[24,25] a dialect of Algol. Compilers for Neliac are available on at least as great a variety of computers as for JOVIAL.

The University of Michigan developed a compiler for a language called Mad, the Michigan Algorithmic Decoder.[26,27] They were quite unhappy at the slow compiling times of Fortran, especially in connection with short problems typical of student use of a computer at a University. Mad was originally programmed for the 704 and has been adapted for the 7090. It too was based on the 1958 version of Algol.

All of these languages derived from Algol 58 are well established, in spite of the fact that the ACM GAMM committee continued its work

and issued its now well known report defining Algol 60.[28]

Algol 60, known simply as Algol, went considerably further than was anticipated in some of the early committee meetings. The language did not limit itself to those areas in which there exists almost universal agreement. Concepts like recursive subroutines, dynamic storage allocation, block structure, own variables and arrays, were introduced which require the inclusion of rather complex structures in the running programs produced by the compiler. Without attempting any serious evaluation of these concepts here, I think it is fair to say that they are difficult, and their inclusion in an Algebraic language that is intended to be universal is controversial. They led to much debate about the difficult areas and tended to obscure some of the more fundamental accomplishments of the Algol committee. Algol set an important precedent in language definition by presenting a rigorous definition of its syntax in the Backus normal form.[29] As compared to Fortran it contains a much more general treatment of iterative loops. It provides a good systematic handling of Boolean expressions and variables and of conditional statements. The most serious deficiency in Algol results from its complete lack of input-output specifications. The handling of input-output is one of the most important services provided by a compiler, and a general purpose Algebraic compiler language is not completely specified until its input-output language has been defined.

Algol compilers have been written for many different computers, but with the exception of Burroughs no computer manufacturer has pushed it very strongly. It is very popular among University and mathematically oriented computer people especially in Europe. For some time in the United States it will probably remain in its status as another available computer language.

4. *Data Processing Compilers.* The largest user by far of data-processing equipment is the United States government. The government, by law and by design, avoids giving preferential treatment to any one computer manufacturer. More than any other computer user, the government is plagued by the problems caused by the lack of compatibility between different kinds of computing equipment, whether manufactured by the same or by different suppliers.

In the spring of 1959 the office of the Secretary of Defense summoned representatives of the major manufacturers and users of data-processing equipment to a meeting in Washington, to discuss the problem associated with the lack of standard programming languages in the data processing area. This was the start of the Committee on Data Systems Languages (CODASYL), that went on to produce COBOL, the common business oriented language. From the beginning their effort was marked by missionary zeal for the cause of English language coding.

Actually, there had been very little previous experience with Data processing compilers. Univac's B-0 or Flow-Matic,[8,9] which was running in 1956, was probably the first true Data-Processing compiler. It introduced the idea of file descriptions, consisting of detailed record and item descriptions, separate from the description of program procedures. It also introduced the idea of using the English language as a programming language.

It is remarkable to note that the Univac I on which Flow-Matic was implemented did not have the data-processing capabilities of a good sized 1401 installation. To add to the problems caused by the inadequacy of the computer, the implementation of the compiler was poor, and compilation was very very slow. There were installations that tried it and dropped it. Others used it, with the philosophy that even with compiling times measured in hours the total output of the installation was greater using the compiler than without it. Experience with Flow-Matic was almost the only experience available on Data Processing compilers prior to the launching of the COBOL project.

A group at IBM had been working for some time on the Commercial Translator,[30] and some early experience on that system was also available.

At the original Defense Department meeting there were two points of view. One group felt that the need was so urgent that it was necessary to work within the state of the art as it

then existed and to specify a common language on that basis as soon as possible. The other group felt that a better understanding of the problems of Data-Processing programming was needed before a standard language could be proposed. They suggested that a longer range approach looking toward the specification of a language in the course of two or three years might produce better results. As a result two committees were set up, a short range committee, and an intermediate range committee. The original charter of the short range committee was to examine existing techniques and languages, and to report back to CODASYL with recommendations as to how these could be used to produce an acceptable language. The committee set to work with a great sense of urgency. A number of companies represented had commitments to produce Data-processing compilers, and representatives of some of these became part of the driving force behind the committee effort. The short range committee decided that the only way it could satisfy its obligations was to start immediately on the design of a new language. The committee became known as the COBOL committee, and their language was COBOL.

Preliminary specifications for the new language were released by the end of 1959, and several companies, Sylvania, RCA, and Univac started almost immediately on implementation on the MOBIDIC, 501, and Univac II respectively.

There then occurred the famous battle of the committees. The intermediate range committee had been meeting occasionally, and on one of these occasions they evaluated the early COBOL specifications and found them wanting. The preliminary specifications for Honeywell's FACT[30] compiler had become available, and the intermediate range committee indicated their feeling that Fact would be a better basis for a Common Business Oriented Language than Cobol.

The COBOL committee had no intention of letting their work up to that time go to waste. With some interesting rhetoric about the course of history having made it impossible to consider any other action, and with the support of the Codasyl executive board, they affirmed Cobol as the Cobol. Of course it needed improvements, but the basic structure would remain. The charter of the Cobol committee was revised to eliminate any reference to short term goals and its effort has continued at an almost unbelievable rate from that time to the present. Computer manufacturers assigned programming Systems people to the committee, essentially on a full time basis. Cobol 60, the first official description of the language, was followed by 61[31] and more recently by 61 extended.[32]

Some manufacturers dragged their feet with respect to Cobol implementation. Cobol was an incomplete and developing language, and some manufacturers, especially Honeywell and IBM, were implementing quite sophisticated data processing compilers of their own which would become obsolete if Cobol were really to achieve its goal. In 1960 the United States government put the full weight of its prestige and purchasing power behind Cobol, and all resistance disappeared. This was accomplished by a simple announcement that the United States government would not purchase or lease computing equipment from any manufacturer unless a Cobol language compiler was available, or unless the manufacturer could prove that the performance of his equipment would not be enhanced by the availability of such a compiler. No such proof was ever attempted for large scale electronic computers.

To evaluate Cobol in this short talk is out of the question. A number of quite good Cobol compilers have been written. The one on the 7090 with which I have had some experience may be typical. It implements only a fraction, less than half I would guess, of the language described in the manual for Cobol 61 extended. No announcement has been made as to whether or when the rest, some of which has only been published very recently, will be implemented. What is there is well done, and does many useful things, but the remaining features are important, as are some that have not yet been put into the manual and which may appear in Cobol 63.

The language is rather clumsy to use; for example, long words like synchronized and computational must be written out all too frequently; but many programmers are willing to put up with this clumsiness because, within its range of applicability the compiler performs

many important functions that would otherwise have to be spelled out in great detail. It is hard to believe that this is the last, or even very close to the last word in data processing languages. Before leaving Data Processing compilers I wish to say a few words about the development of the FACT compiler.

In 1958 Honeywell, after almost leaving the computer business because of the failure of their Datamatic 1000 computer, decided to make an all out effort to capture part of the medium priced computer market with their Honeywell 800 computer. The computer itself is very interesting but that is part of another talk.

They started a trend, now well established, of contracting out their programming systems development, contracting with Computer Usage Co. for a Fortran language compiler.

Most interesting from our point of view was their effort in the Data Processing field. On the basis of a contract with Honeywell, the Computer Sciences Corporation was organized. Their contract called for the design and production of a Data processing compiler they called FACT.[30,33]

Fact combined the ideas of data processing generators as developed by Univac, GE Hanford,[34] Surge[35] and 9PAC with the concepts of English language data processing compilers that had been developed in connection with Univac's Flow-Matic and IBM's commercial translator.

The result was a very powerful and very interesting compiler. When completed it contained over 250,000 three address instructions. Designed to work on configurations as small as 4096 words of storage and 4 tape units it was not as fast as some more recent compilers on larger machines.

The Fact design went far beyond the original COBOL specifications,[30] and has had considerable influence on the later COBOL development.

Like all other manufacturers Honeywell has decided to go along with the COBOL language, and Fact will probably fall into disuse.

5. *Assemblers and Operating Systems.* Symbolic assembly language has become an almost universal form for addressing a computer in a computer oriented language.

After the first 704's were delivered in 1956 a number of users produced assembly routines for use on the computer. One of the early standardization efforts involved a choice of a standard assembly program to be used by Share, the 704 users group. It is a sign of some of the thinking that was current then that the standard chosen was UASAP.[36] The first SAP was a very basic assembly system. It did practically nothing but one-to-one translation, and left the programmer in complete control of all of the parameters of the program. In the early days many users felt that this was all an assembly system should do. Some still feel that way, but on most computers the simple assembly system has been replaced by the full scale computer oriented compiler in which one-to-one code translation is augmented by extensive libraries of subroutines and generators and by allocation, segmentation, and other program-organization features.[37]

The word-macro-instruction apparently was coined in connection with the symbolic assembly systems that were developed for IBM's 702/705 computers. These Autocoder[38] systems with their large macro-instruction libraries have been used for huge amounts of data processing programming on a number of machines.

Assembly systems gradually grew into or became incorporated into operating systems.[39,40] Perhaps the earliest monitor system on the 704 was put into operation at the General Motors Research center.[41,42] The idea of automatic sequencing of batches of jobs spread rapidly until it was almost universally used in connection with large computers. It made it possible for large computers to handle small jobs with reasonable efficiency and greatly extended their range of application. The idea of such systems was to run the computer without any stops, and to relegate the operator to occasional mounting of tapes, and otherwise to responding to very simple requests presented to him on the on-line printer. Under such a system debugging becomes a completely off-line process. The only response to trouble in a program is to dump and get on with the next program.

At the end of 1956 IBM announced its new 709 computer. The 709 was essentially a 704 with internally buffered input and output.

As mentioned earlier, IBM was at its peak of penetration of the large scale scientific computer market at that time, and the rest of the industry watched with great interest as many of the best programming systems people representing many leading scientific computer installations met as a Share committee to design the very complex programming system which was eventually called SOS (Share Operating System).

The design of programming systems by large committees representing many companies and institutions has almost invariably led to disappointing results. SOS was no exception. Planned mainly by personnel of West Coast aircraft and research companies, it was to be written according to their specifications by the IBM programming systems activity on the East Coast. Separation of design and implementation responsibility by 3000 miles is almost enough in itself to guarantee great difficulty, if not complete failure. In 1958 the chairman of the Share 709 system committee wrôte,[43] "The fundamental procedures used throughout the system will undoubtedly be retained in every installation." This has not been the case. The SOS system is now in use at only a very few installations. There are many reasons, of which I would like to suggest just a few. SOS put all of its emphasis on the computer oriented programming system. The time during which SOS was being designed and implemented was the time during which the attitude toward Fortran was changing from polite skepticism to very general acceptance. By the time SOS was in nearly full operation some installations were using almost nothing but Fortran. Apparently little or no effort had been expended on the problem of compatibility between SOS and Fortran. It was only in 1962 that an SOS system which handles Fortran was distributed by the Rand Corporation.[44] Their system accepts Fortran source programs, and produces the binary symbolic or squoze decks that can be combined with other programs produced by the SOS system. IBM boasted of over 50 man years of effort on the SOS system for the 709. They spent almost no effort on Fortran for the 709, on the theory that Fortran was developed for the 704 would be adequate. The Fortran II system that was originally distributed for the 709 took no advantage of the fact that the 709 hardware permitted buffered input and output. The SOS system provided a very elaborate buffering system.

SOS proposed to provide a system in which the programmer would need to know and use only one language, the compiler source language. One of its major achievements was the provision of source language modification of programs at load time without full recompilation. A very versatile debugging system was built around this feature. While this and other features of the system are extremely attractive, there is a serious question as to whether they are worth the price paid in system complexity, and in increased loading time. I think it is interesting to point out that a relatively simple assembly system, FAP, and a very basic operating system, the Fortran Monitor System, both originated at customer installations and not by the manufacturer, became the most widely used systems on the 709/90 computers. Quite similar systems were introduced on competitive equipment, the Philco 2000 and the CDC 1604. Complexity and system rigidity no doubt contributed to the fact that SOS was not generally accepted. It will be interesting to follow the history of a new and very complicated system, the IBSYS/IBJOB complex that has recently been introduced by IBM on the 7090 and related machines. A critique of these systems is far beyond the scope of this discussion. A few comments may be in order. IBJOB presents a very elaborate assembly system MAP, and translators from two languages, FORTRAN IV and Cobol into Map. They are then translated into a language that is close to machine language, with the final steps of the translation left to a very complicated loader. The design, which calls for the translation of problem oriented source languages into an intermediate computer oriented source language is very attractive. By having the assembly system do most of the work of the compiler it is possible to have many of the features of the problem oriented language available by means of subroutine calls to those who prefer to write in assembly language. This design philosophy is attractive, but I think that it is wrong. Attractiveness and elegance should not be the determining design criteria for production compiling systems. Design of a system

is a compromise between many design criteria. One of the most important is keeping the system overhead cost low on the many problems that do not require the use of very sophisticated system features. The code produced by a compiler like Fortran is straightforward simple code. The assembler for such code can be simple and straightforward. The loading program can be designed to permit easy combination with programs produced by other systems. An assembler designed to aid in the production of large programming systems contains many features that are seldom used except in the coding of such systems. A wasteful mismatch may occur when the output of Fortran is fed through such an assembler.

Not so very many years ago there was quite a bit of discussion as to whether general purpose operating systems should be designed and supplied by the manufacturers. Some users felt that the very great difference in the job mix and operating philosophy at the various installations called for specially designed and tailored system programs. For a time the argument seem to be settled by the almost universal assumption that operating systems, and computer software in general were as much an obligation of the manufacturer as was the building of the computers themselves. I wonder if this assumption will be able to stand up in face of the rapid developments in the large computer field that will lead to computing systems that are very much more diverse, and very much more complex than those that are in general use today.

In the large computer field multiprocessing and multiprogramming systems will soon become the rule rather than the exception. Many experiments in these directions are being tried with computers of the generation that is now coming to an end. Systems combining a 7094, a 1301 Disc unit and a 7040 will soon be commonplace. There are a number of military systems involving several computers and much peripheral equipment all working together under a common operating system.

Among newer computers already delivered to customers there are several models that have been designed to make it possible and practical to run peripheral equipment on-line while simultaneously carrying out independent computer processing tasks. The distinction between off-line and on-line tends to disappear on such systems, and the operating systems must be able to control equipment in many different modes. Systems already delivered that have some features that permit multiprogramming and multiprocessing include the Honeywell 800, The Ferranti Atlas, The Burroughs 5000 and D825. There is some very interesting recent literature about programming systems for these computers.[45,46,47]

In the next generation of large computers it may be possible to implement true demand processing systems. Demand systems have been advocated by many in opposition to batching systems. In a demand system problems are submitted as they arise. The system controls the input of jobs and the scheduling of jobs by stacking jobs in queues according to length, priority, etc. A demand system requires multiprogramming facilities, but also requires much more elaborate decision making on the part of an executive system than is present in most monitors today.

The complexity required in some of these operating systems may seem to require that they be uniform systems designed and produced by the manufacturer. But, another feature that is being stressed more and more is modularity, which permits an almost unlimited variety in system configurations. It is very difficult to design a single operating system that is appropriate for a computing system based on Disc storage, and also for one based on tapes or drums, and also for any combination of auxiliary devices. The problem will get more complicated when high speed storage at different levels is available in various quantities. It is quite reasonable to anticipate a system in the next few years that will have a very high speed film memory, backed up by a fast core memory, backed up by a large and somewhat slower core memory, backed up by high speed drums, then discs and tapes. It will be a real challenge to design programming systems that are valid for all combinations in such systems.

In the early days one of the aims of the operating system was to get the human being out of the system as much as possible. In a multi-programming system it is possible to allow human

intervention in the running of a program without any appreciable loss of computer time, since the computer will presumably have other programs it can work on. There has been a great deal of publicity given to the experiments in the use of on-line consoles on present day systems.[48] Such consoles may be a routine part of many computing systems in a few years.

In recent issues of a number of Computer publications there is an advertisement for a computer that claims to be faster than the 7090 and costs only $215,000 dollars. Whether or not the claim is true, it does serve to emphasize the fact that the cost of computer processing capability is going to go down rapidly. It is going to be economically feasible to put together extremely large, varied, and complicated concentrations of computer components. Programming systems are going to increase in number and complexity, and the job of the system program designer is going to remain as it always has been, very difficult, but very, very interesting.

6. *Bibliography.* This is not intended to be a complete bibliography of the field of Programming Systems and Languages. It is rather a selected list of publications that may help to document the text. A great deal of reference material is contained in manuals published by computer manufacturers. It would serve no useful purpose to list such manuals here. Manuals exist for just about every programming system mentioned in the text, and the mention of a language or system may be interpreted as an implied reference to the manual. I have attempted to include specific references to sources other than manuals for most systems discussed, but in many cases the system manuals provide better and more complete documentation.

In the following Bibliography the abbreviation "1954 Symposium" will be used to refer to the Proceedings of a Symposium on Automatic Programming for Digital Computers, Navy Mathematical Computing Advisory Panel, Office of Naval Research, Washington, D. C. May 13-14, 1954. These proceedings include a bibliography on pp. 150-152.

Additional references will be found in the bibliography on pages 260-270 of reference 49.

1. AIKEN, H. H. and HOPPER, G. M. The Automatic Sequence Controlled Calculator. Electrical Engineering 65 (1946) pp. 384-391, 449-454, 522-528.
2. WILKES, M. V., WHEELER, D. J. and GILL, S. The Preparation of Programs for an Electronic Digital Computer. Addison-Wesley 1957. (First edition published in 1951.)
3. MACMILLAN, D. B. and STARK, R. H. "Floating Decimal" Calculation on the IBM Card Programmed Electronic Calculator. Mathematical Tables and other Aids to Computation. (Mathematics of Computation.) 5(1951) pp. 86-92.
4. ADAMS, CHARLES W. and LANING, J. H. JR. The M.I.T. System of Automatic Coding: Comprehensive, Summer Session, and Algebraic. 1954 Symposium. pp. 40-68.
5. BENNINGTON, H. D. Production of Large Computer Programs. Proceedings of a Symposium on Advanced Programming Methods for Digital Computers. Navy Mathematical Computing Advisory Panel and Office of Naval Research. Washington, D. C. June 28, 29, 1956.
6. HOPPER, G. M. The Education of a Computer. Proceedings of the Conference of the ACM. Pittsburgh, 1952.
7. RIDGWAY, R. K. Compiling Routines. Proceedings of the Conference of the ACM Toronto, 1952.
8. TAYLOR, A. E. The Flow-Matic and Math-Matic Automatic Programming Systems. Annual Review of Automatic Programming 1 (1960) pp. 196-206. Pergamon.
9. KINZLER, H. and MOSKOWITZ, P. M. The Procedure Translator—A System of Automatic Programming. Automatic Coding. Monograph No. 3. Journal of the Franklin Institute. Philadelphia, 1957 pp. 39-55.
10. HOLBERTON, F. E. Application of Automatic Coding to Logical Processes. 1954 Symposium pp. 34-39.
11. KAHRIMANIAN, H. G. Analytic Differentiation by a Digital Computer. 1954 Symposium pp. 6-14.
12. HOLT, ANATOL W. General Purpose Programming Systems. Communications of the ACM 1 (1958) pp. 7-9.

13. BACKUS, J. W. and HERRICK, H. IBM 701 Speedcoding and other Automatic Programming Systems. 1954 Symposium. pp. 106-113.
14. BACKUS, J. W. The IBM 701 Speedcoding System. Journal of the ACM 1(1954) 4-6.
15. BAKER, CHARLES L. The Pact I Coding System for the IBM Type 701. Journal of the ACM 3 (1956) pp. 272-278. This is one of seven papers on the Pact I system in the same issue of JACM.
16. STEEL, T. B. JR. Pact 1A. Journal of the ACM. 4 (1957) pp. 8-11.
17. WOLONTIS, V. M. A Complete Floating Decimal Interpretive System. Technical Newsletter No. 11 IBM Applied Science Division. 1956.
18. PERLIS, A. J. and SMITH, J. W. A Mathematical Language Compiler. Automatic Coding. Monograph No. 3. Journal of the Franklin Institute. Phildelphia 1957 pp. 87-102.
19. BACKUS, J. W. et al. The Fortran Automatic Coding System. Proceedings of the Western Joint Computer Conference. Los Angeles 1957 pp. 188-198.
20. SHERIDAN, P. B. The Arithmetic Translator-Compiler of the IBM Fortran Automatic Coding System. Communications of the ACM 2 (February 1959) pp. 9-21.
21. PERLIS, A. J. and SAMELSON, K. Preliminary Report — International Algebraic Language. Communications of the ACM 1 (December 1958) pp. 8-22.
22. ROSEN, S. and GOLDBERY, I. B. Altac, The Transac Algebraic Translator. Preprints of papers presented at the 14th National Conference of the ACM. Boston, 1959.
22. A ROSEN, S. Altac, Fortran, and Compatability. Preprints of papers presented at the 16th National Conference of the ACM, Los Angeles, 1961.
23. SHAW, C. J. Jovial—A Programming Language for Real-time Command Systems. Annual Review of Automatic Programming 3(1963) pp. 53-119. Pergamon.
24. HUSKEY, H. D., HALSTEAD, M. H. and MCARTHUR, R. Neliac, A Dialect of Algol. Communicaitons of the ACM 3(1960) pp. 463-468.
25. HALSTEAD, M. H. Machine Independent Computer Programming. Spartan Books, 1962.
26. ARDEN, B. W., GALLER, B. A., GRAHAM, R. M. The Internal Organization of the Mad Translator. Communication of the ACM 4(1961). This issue of the CACM contains the Proceedings of an ACM Compiler Symposium, Washington D.C. November 17-18, 1960.
27. GALLER, B. A. The Language of Computers. McGraw Hill, 1962.
28. NAUR, P. Editor. Revised Report on the Algorithmic Language Algol 60. Communications of the ACM 6(1963) pp. 1-17. Also published in Numerische Mathematic and the Computer Journal.
29. BACKUS, J. W. The Syntax and Semantics of the Proposed International Algebraic Language of the Zurich ACM-GAMM Conference. Information Processing. Proceedings of ICIP, UNESCO, Paris 1959 pp. 125-132.
30. CLIPPINGER, R. F. FACT A Business Compiler. Description and Comparison with COBOL and Commercial Translator. Annual Review in Automatic Programming. Vol. 2(1961) Pergamon. pp. 231-292.
31. SAMMET, JEAN E. Basic Elements of COBOL 61. Communications of the ACM 5(1962) p. 237-253.
32. COBOL-1961 Extended. External Specifications for a Common Business Oriented Language Dept. of Defense. 1962. U.S. Govt Printing Office. Washington, D.C.
33. CLIPPINGER, R. F. FACT. The Computer Journal 5(1962) pp. 112-119.
34. MCGEE, W. C. Generalization: Key to Successful Electronic Data Processing Journal of the ACM 6(1959) pp. 1-23.
35. LONGO, F. SURGE: A Recording of the COBOL Merchandise Control Algorithm. Communications of the ACM 5(1962) pp. 98-100.
36. MELCHER, W. P. Share Assembler UASAP 3-7. Share Distributor 564. 1958. The Original UASAP 1 was written by Mr. Roy Nutt.

37. CHEATHAM JR., T. E., and LEONARD, G. F. An introduction to the CL-II Programming System. Document CAD-63-4-SD. Computer Associates Inc. Wakefield, Mass. 1963.

38. GOLDFINGER, R. The IBM Type 705 Autocoder Proceedings of the Western Joint Computer Conference. San Francisco 1956.

39. ORCHARD-HAYS, W. The Evolution of Programming Systems. Proceedings of the IRE 49(1961) pp. 283-295.

40. MEALY, G. H. Operating Systems. Rand Corporation Document P-2584. Santa Monica, California, 1962.

41. RYCKMAN, G. F. The Computer Operation Language. Proceedings of the Western Joint Computer Conference 1960, pp. 341-343.

42. FISHMAN, J. J., HARROFF, D. F., LIVERMORE, F. G. and KUHN, E. F System (FS-3). Internal Publication of the General Motors Research Laboratories (1960).

43. SHELL, D. L. The Share 709 System: A Cooperative Effort. Journal of the ACM 6(1959) p. 123-127. This is one of the six papers on the Share 709 System in the same issue of JACM.

44. BRYAN, G. E., Editor. The Rand-Share Operating System Manual for the IBM 7090 Computer. Memorandum RM-3327-PR, Rand Corporation. Sept. 1962.

45. KILBURN, T., HOWARTH, D. J., PAYNE, R. B. and SUMNER, F. H. The Manchester University Atlas Operating System. Parts 1 and II. The Computer Journal 4(1961) pp. 222-229.

46. KILBURN, T., PAYNE, R. B. and HOWARTH, D. J. The Atlas Supervisor. Proceedings of the Eastern Joint Computer Conference, Washington, D. C. 1961. pp. 279-294.

47. THOMPSON, R. N. and WILKINSON, J. A. The D825 Automatic Operating and Scheduling Program. Proceedings of The Spring Joint Computer Conference. Detroit, Mich. 1963 pp. 41-49.

48. CORBATO, F. J., MERWIN-DAGGETT, M. and DALEY, R. C. An Experimental Time-Sharing System. Proceedings of The Spring Joint Computer Conference. San Francisco, 1962.

49. CARR, J. W., III. Programming and Coding. Part B of Handbook of Automation, Computation and Control. Volume 2. Wiley 1959.

BOUNDED CONTEXT TRANSLATION

Robert M. Graham
Computation Center
Massachusetts Institute of Technology
Cambridge, Mass.

All translators are syntax-directed in the sense that the translator must obviously recognize the various syntactic structures and the output of the translator is a function of the syntax of the language. The term syntax-directed is usually applied to a translator which contains a direct encoding of the syntax of the language, this direct encoding being used by the translator as data. The companion paper by Cheatham and Sattley is concerned with this type of translation.[1] In the other class of translators the syntax is essentially buried in the coding of the translator. Most of the algebraic languages in use are precedence grammars,[2,3] or close enough so that the properties of precedence grammars are useful. Using the properties of precedence grammars, bounded context translation is possible. At each step in the scan of an expression in bounded context translation the decision as to what action to take next is a function of the symbol currently under scan and of N symbols on either side (where N is fixed for the particular language).

Most translators produce an intermediate form of the program before producing the final machine code. Once this intermediate form is produced, the distinction between syntax-directed translation and other methods disappears. The primary goal of the intermediate form is to encode the program in a form which is easily and efficiently processed by the computer. Most optimization algorithms, such as elimination of common subexpressions and optimum evaluation of Boolean expressions,[8] are much simpler when applied to some intermediate form rather than to the original expression or the final machine language version. The intermediate form exhibits the structure (i.e., which subexpressions are the operands of each operator) and the order of evaluation of the subexpressions.

In this paper we will restrict ourselves to the source language defined in Appendix A. For the sake of brevity when we say expression we will mean either an expression, as defined in Appendix A, or an assignment statement. This language has no constants and identifiers are single letters, thus avoiding the problem of recognition. A translator for a language such as ALGOL would have a recognizer which would scan a statement and which, each time it was called, would recognize the next element in the input string. After recognizing an identifier the recognizer would store it in a symbol table if it were not already there. The symbol table would also contain other information pertaining to the identifier such as type, address, dimension, etc. The recognizer would then return a pointer to the place in the symbol table where that identifier and its properties were stored. A constant, after recognition, would be stored in a table of constants, given an internal identifier, and treated from that point on just as any other identifier would be. When an operator was recognized, a pointer to the place

in the operator table where the properties of that operator were stored would be returned. Thus we see that the recognizer would effectively reduce an expression of a language such as ALGOL to an expression of our language; that is, each element would be reduced to a single symbol. In general, the intermediate form does not contain the identifiers or operators but the pointers supplied by the recognizer. However, for clarity, in the examples we will use the actual identifiers and operators. The recognizer is heavily dependent upon the details of the particular language being translated and such characteristics of the computer as character representation, word size, etc. The algorithms to be discussed in this paper depend only upon a few very general properties of the source language. Most of the common compiler languages have these properties so the techniques discussed here are essentially language independent. While the algorithms which we will describe here apply only to the simple expressions of Appendix A, several of them may be trivially generalized to translate more complex expressions and even the bulk of the ALGOL statements.[4,8,12]

Some representation of the tree form of an expression is used in several translators. Figure 1 shows the tree form of the expression A:=B*C+D*(E—F) and the matrix representation of the tree[5,6,7,8] The first column is the operator, the second column is the left hand operand, and the third column is the right-hand operand. An integer represents the result of the row with that number. The subexpressions are to be evaluated in the sequence in which the rows are written. Figure 2 shows the tree form used by Ross.[12] The first three columns in the representation are the same as in Figure 1. There are two additional columns which explicitly state the sequence of subexpression evaluation. The fourth column is the minor evaluation string pointer (dashed arrows in the tree) and the fifth column is the major evaluation string pointer (solid arrows in the tree). In the example the evaluation is to start with row 2. The rules for following the evaluation strings for the row under examination are:

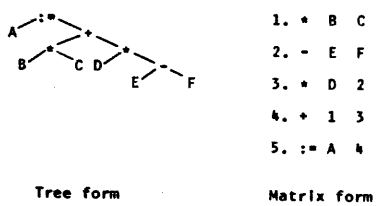


Figure 1.

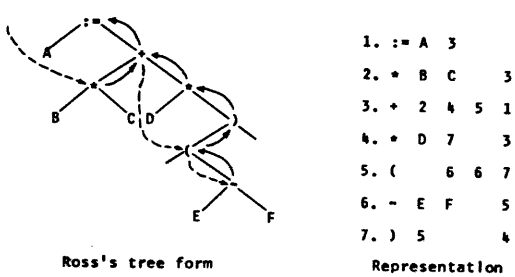


Figure 2.

1. If this is the first time this row has been examined and,
   a. If the minor string pointer is not empty; proceed to the row named by it.
   b. If the minor string pointer is empty; evaluate this row and proceed to the row named by the major string pointer.
2. If this is the second time this row has been examined, evaluate this row and proceed to the row named by the major string pointer.

In this example the '('and')' symbols should be treated as "do nothing" operators when their evaluation is called for by the rules. Ross's representation is part of a more general system and, hence, some features of the representation are not pertinent to this discussion. A representation similar to Ross's based on threaded lists is described in Ref. 9.

Evans[4] bases his intermediate form on postfix notation. The expression A:=B*C+D*(E—F) in postfix form is ABC*DEF—*+:=. In this form both the structure and the order of subexpression evaluation are implicit. The right-hand operand of an operator is the first complete expression to the left of the operator and

the left-hand operand is the second complete expression to the left of the operator. In the example the right-hand operand of the '+' is 'DEF—*' and the left hand operand is 'BC*' The subexpressions are evaluated in left to right order.

The transformation of a completely parenthesized expression into matrix form is relatively simple. The expression is scanned from left to right. The following rules are applied to each symbol as it is encountered (a variable name is considered as a single symbol):

1. If the symbol is not a ')'; continue the scan.
2. If the symbol is a ')' and the expression is properly formed then the four symbols to the left of the ')' should be of the form 's1#s2', where 's1' and 's2' stand for variable names or integers (row numbers) and '#' is any operator; write '# s1 s2' as the next row of the matrix and replace '(s1#2)' in the expression by the number of the row just written in the matrix. Continue the scan.

In Figure 3 the changes in the expression are shown on the left, the small arrow under the expression indicating the points, during the scan, at which a row is written in the matrix. The row actually written at that point is shown on the right.

Completely parenthesized expressions have such an overabundance of parentheses that they are difficult to read; hence, languages such as ALGOL have precedence rules making it unnecessary to write completely parenthesized expressions. The language is a precedence grammar if the precedence rules are such that given two adjacent operators it is unambiguous which is to be evaluated first. Thus if a language is a precedence grammar it is possible to construct

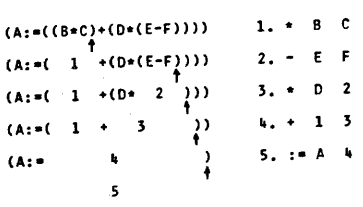


Figure 3.

	+	-	*	/
+	left	left	right	right
-	left	left	right	right
*	left	left	left	left
/	left	left	left	left
:=	right	right	right	right

Figure 4.

a table of the type shown in Figure 4. To determine which of two adjacent operators to evaluate first, find the intersection of the row labeled with the left operator and the column labeled with the right operator. In the context of transforming an expression into matrix form, the order of evaluation of the operators is to be interpreted as the order of their entry into the matrix. A subexpression enclosed by parentheses is to be completely evaluated before any consideration is given to the operators on either side of it. Applying these considerations we have the following rules for transforming an expression to matrix form. We enclose the expression on the left by '|—' and on the right by '—|'. Again we scan from left to right, applying the rules to each symbol in turn:

1. If the symbol is an operator, #1, and the left context is,
   a. '|—s1'; continue the scan.
   b. '(s1'; continue the scan.
   c. 's2#2s1'; look up #2#1 in the table. If the table entry is,
      i. 'right'; continue the scan.
      ii. 'left'; write '#2 s2 s1' as the next row of the matrix, replace 's2#2s1' in the expression by the number of the row just written in the matrix, and apply rule 1 again.
2. If the symbol is ')' and the left context is,
   a. 's2#s1'; write '# s2 s1' as the next row of the matrix, replace 's2#s1' in

the expression by the number of the row just written in the matrix, and apply rule 2 again.

b. '(s'; replace '(s)' by 's' and continue the scan.

3. If the symbol is '—|' and the left context is,

a. 's2#s1'; write '# s2 s1' as the next row of the matrix, replace 's2#s1' in the expression by the number of the row just written in the matrix, and apply rule 3 again.

b. '|— s'; the expression has been completely transformed into matrix form.

This is the essence of bounded context translation. The rules just stated show that N=3 for the precedence grammar of appendix A. That is, in deciding what action to take next, at most only the three symbols immediately to the left of the symbol currently under scan need be examined regardless of the length of the expression being translated.

Before examining some ways that bounded context analysis has actually been implemented let us restate, precisely, the above rules in the form of a flow chart (Figure 5). In the flow charts of this paper the true exit of a decision box is marked t and the false exit is marked f. We consider the expression being analyzed as a string of symbols, S (indexed by k), bounded on the left by '|—' and on the right by '—|'. We will distinguish three classes of symbols:

1. I, the class of identifiers: any variable name.
2. R, the class of matrix row numbers: any integer.
3. θ, the class of operator symbols: $\theta = \{+, -, *, /, :=, (,), |-, -|\}$. For future use we distinguish the subclass of arithmetic operators, $\theta^\circ = \{+, -, *, /\}$.

Symbols from the input string are transferred to an auxiliary list, L (indexed by j). This will avoid the problem of gaps when replacements are made in the input string. M is the matrix (with i the row index) and T(x, y) is a function whose arguments are a pair of operators and whose value is the label found at the intersection of the x-row and the y-column in Figure 6. The label ERR has been filled in for all illegal operator pairs. If one of these occurs then the expression is incorrectly formed. The questions of the detection of incorrectly formed expressions and what action to take when errors are detected is very important in actual translator construction. These questions will not, however, be discussed in this paper. We will assume here that all expressions are properly formed. When the function T(x, y) takes on the value END then the matrix form of the expression is complete.

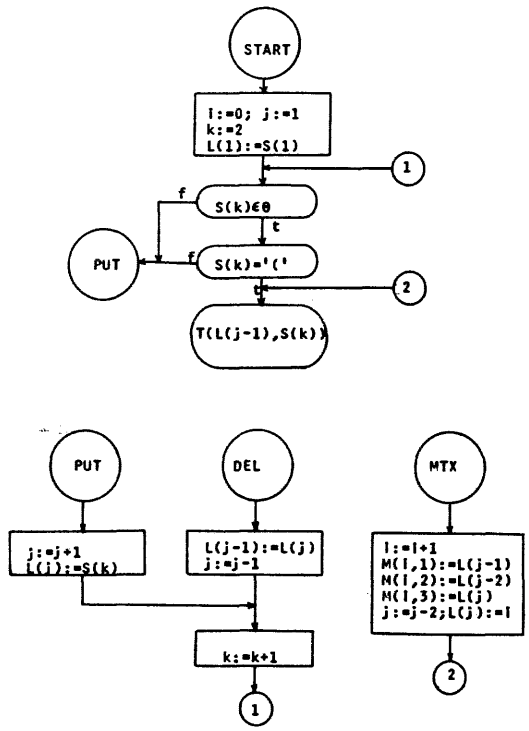


Figure 5.

	+	-	*	/	:=	)	⊣
+	MTX	MTX	PUT	PUT	ERR	MTX	MTX
-	MTX	MTX	PUT	PUT	ERR	MTX	MTX
*	MTX	MTX	MTX	MTX	ERR	MTX	MTX
/	MTX	MTX	MTX	MTX	ERR	MTX	MTX
:=	PUT	PUT	PUT	PUT	ERR	ERR	MTX
(	PUT	PUT	PUT	PUT	ERR	DEL	ERR
⊢	PUT	PUT	PUT	PUT	PUT	ERR	END

Figure 6.

It is obvious from the table in Figure 6 that most operator pairs cause one of two actions to take place: 1) the operator currently under scan is put on the list, or 2) rows are written in the matrix until an operator pair is found which calls for some other action. The only exceptions are the removal of parentheses and termination of the transformation. If a precedence function is defined for each operator then a table of all operator pairs is not necessary. A precedence function, P(x), is defined such that, given a pair of operators #1#2, if P(#1)#≧#P(#2) then the operator #1 is evaluated before the operator #2. A precedence function for the operators used in the previous examples is defined in Figure 7. The algorithm for transforming an expression into matrix form that is used in GAT[5] and MAD[6,7] uses a precedence function. The flow chart in Figure 8 gives an algorithm for generating the matrix form of the expression. This algorithm differs from the GAT-MAD algorithm only in the direction of scanning the expression. Notice that, assuming all expressions are properly formed and since P('(') < P(')') and P('|—') < P('—|'), when the test P(Lj—1)) ≧ P(S(k))fails then S(k)=')' implies L(j—1) = '(' and S(k) = '—|' implies L(j—1) = '|—'.

Bauer and Samelson[10] use two lists, one, L, in the translator and one, N, in the object program. L, called the "symbols cellar," is used for storing operators which can not be evaluated yet, just as in the previous algorithms. N, called the "numbers cellar," is used by the object program during execution for the temporary storage of partial results and values of identifiers. Their algorithm does not generate an intermediate form, but immediately generates machine code. In the examples of this paper, the machine language used will be that described in Appendix B. This language is very similar to the FAP symbolic machine language used on the IBM 709–90–94 computers. C (indexed by m) will be a matrix where generated machine instructions are stored. The first column will contain the operation code and the second column the address. Figure 9 is the table, for Bauer and Samelson's algorithm, corresponding to the table in Figure 6, and Figure 10 is their algorithm. Whenever an identifier is encountered in the scan, instructions are generated to move its value onto the N list. Bauer and Samelson give, in the same paper, modifications to this algorithm which will generate more efficient machine code.

x	P(x)
/	7
*	7
+	6
-	6
:=	5
)	4
(	3
⊣	2
⊢	1

Figure 7.

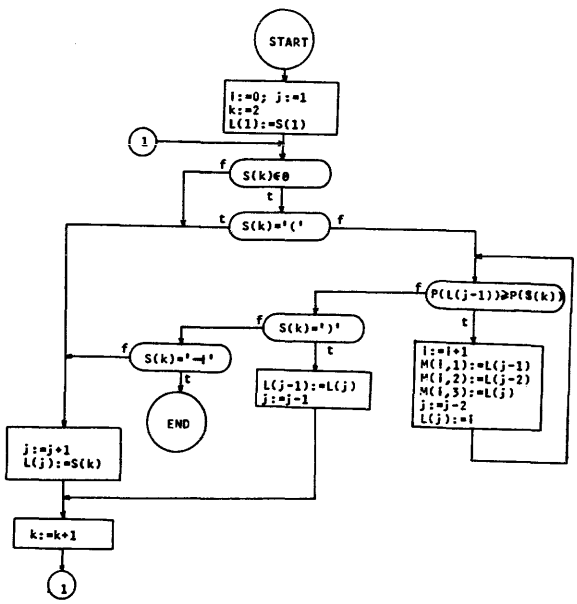


Figure 8.

	+	-	*	/	(	)	⊣
⊢	PUT	PUT	PUT	PUT	PUT	ERR	END
+	MTX1	MTX1	PUT	PUT	PUT	MTX1	MTX1
-	MTX2	MTX2	PUT	PUT	PUT	MTX2	MTX2
*	MTX3	MTX3	MTX3	MTX3	PUT	MTX3	MTX3
/	MTX4	MTX4	MTX4	MTX4	PUT	MTX4	MTX4
(	PUT	PUT	PUT	PUT	PUT	DEL	ERR

Figure 9.

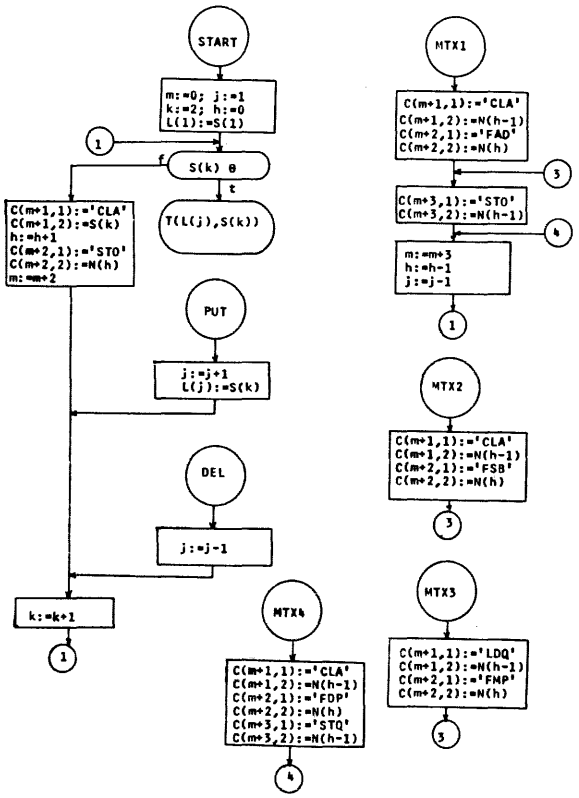


Figure 10.

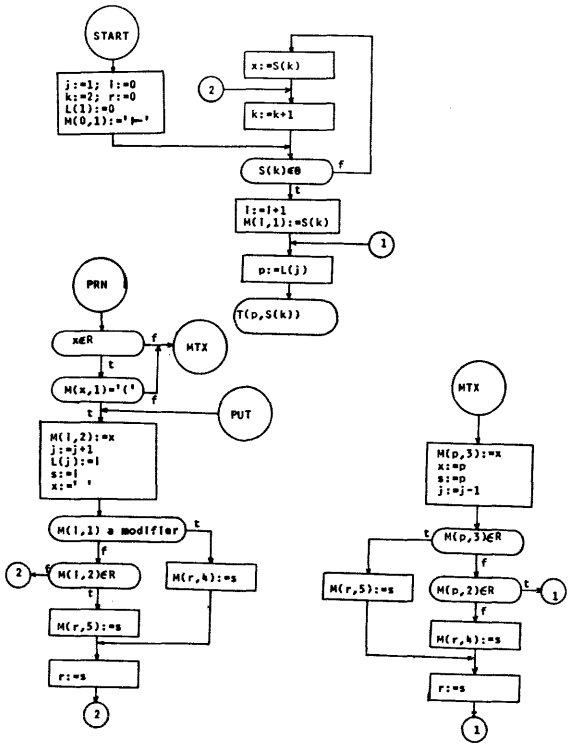


Figure 11.

	+	-	*	/	:=	(	)	⊣
+	MTX	MTX	PUT	PUT	ERR	PUT	PRN	MTX
-	MTX	MTX	PUT	PUT	ERR	PUT	PRN	MTX*
*	MTX	MTX	MTX	MTX	ERR	PUT	PRN	MTX
/	MTX	MTX	MTX	MTX	ERR	PUT	PRN	MTX
:=	PUT	PUT	PUT	PUT	ERR	PUT	ERR	MTX
(	PUT	PUT	PUT	PUT	ERR	PUT	MTX	ERR
)	MTX	MTX	MTX	MTX	ERR	ERR	ERR	MTX
⊢	PUT	PUT	PUT	PUT	PUT	PUT	ERR	END

Figure 12.

Ross's algorithm, as given in Ref. 12, transforms more than just expressions into tree form. In the version of his algorithm given in Figures 11 and 12 the machinery not needed to transform expressions has been omitted. A minor evaluation string pointer is set whenever the right operand of an operator is set and both operands are variables, or whenever a left operand is set and the operator is a modifier. The only modifier in an expression is the '('. The minor evaluation string is forced to pass through modifiers since a modifier may change the interpretation of the right operand. For example, the right operand of '(' may be either a normal subexpression or the argument of a function depending upon whether the left argument of the '(' is empty or is an identifier. A major evaulation string pointer is set whenever a right or left operand is set and the operand is a row number.

Evans[4] uses an algorithm based on Floyd's productions.[11] Instead of generating machine code directly as Floyd did in his paper, Evans transforms the expression into postfix form. This algorithm is probably the most versatile of the lagorithms which we have described here. The central idea here is a set of productions which determine the course of action at each step in the translation process. A production consists of five parts;

1. A top of stack configuration.
2. A replacement for the configuration of part 1.
3. The name of an action routine.
4. Scan, no scan flag.
5. Which production to consider next.

Figure 14 gives the productions for transforming an expression into postfix form. The expression is scanned from left to right. As each new character is scanned it is put on the top of the pushdown stack and the productions are then searched for the first one whose part 1 matches the present top of the stack (when a class symbol such as θ° appears, any member of that class will give a match). When a match is found that portion of the top of the stack which matched part 1 of the production is replaced by part 2 of the production. If part 2 is empty, the replacement degenerates to the deletion of the symbols, which matched part 1, from the top of the stack. The action routine named in part 3 is then executed. After the action routine has been executed the productions are again searched for a match with the top of the stack; however, the search is started with the production whose line number is given in part 5 of the last interpreted production. If

	1	2	3	4	5
1	I		OUT	*	3
2	(	(	NOP	*	1
3	θ°	θ°	COMP	*	1
4	:=	:=	OUTP('loc')	*	1
5	-I	-I	COMP		END
6	)	)	COMP		7
7	()		NOP	*	3

P-Table

Figure 13.

x	P(x)
*	7
/	7
+	6
-	6
:=	5
)	4
-I	3
(	2
I-	1

Figure 14.

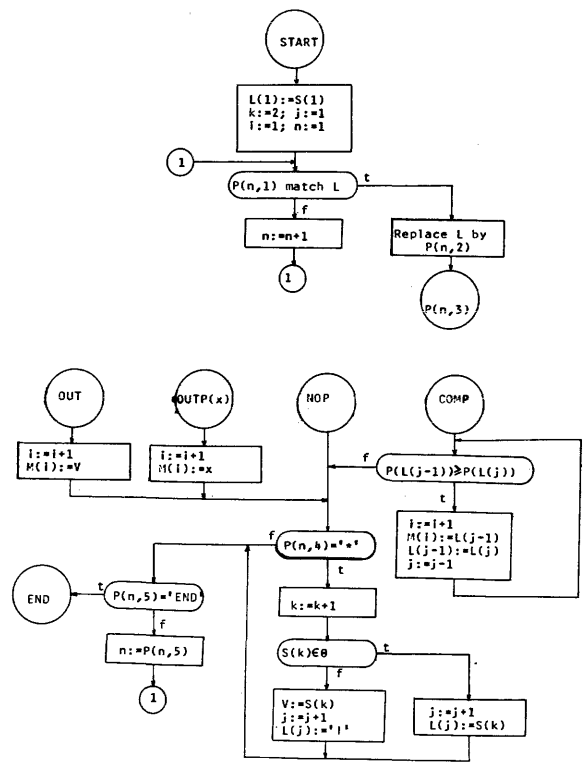


Figure 15.

a '*' appears in part 4, a new symbol is scanned from the input string before the search continues. The productions given in Figure 13 are sufficient only to transform the simple expressions of this paper. The productions that Evans gives will transform all the statements of ALGOL into postfix form. Figure 15 gives an algorithm for transforming an expression into postfix form using the productions of Figure 13. The action routine OUT puts the last identifier scanned into the output string, M. The action routine OUTP(x) puts its argument, x, into the output string. The productions cause the unary operator 'loc' to be introduced into the output string following a variable which is on the left side of ':=', which indicates that a location (where a value is to be stored) is involved rather than a value. The action routine COMP uses the precedence function, defined in Figure 14, to determine when operators are placed in the output string (i.e., to determine the sequence of evaluation of the operators).

Once the matrix form of the expression has been generated, the final translation to symbolic machine code is relatively simple. Corresponding to each operator is a set of machine instructions, a pattern. Figure 16 gives the patterns for the operators of our language; the first column is the operation code and the second column is the address. The matrix is translated one row at a time, in sequence. Row i of

	1	2	operator
1	CLA	1	+
2	FAD	r	
3	STO	t	
4	CLA	1	-
5	FSB	r	
6	STO	t	
7	LDQ	1	*
8	FMP	r	
9	STO	t	
10	CLA	1	/
11	FDP	r	
12	STQ	t	
13	CLA	r	:=
14	STO	1	

Pattern Table

Figure 16.

the matrix, M, is translated by making a copy, in the code output matrix, of the pattern corresponding to the operator M(i, 1), replacing all occurrences of '1' by the left operand, M(i, 2), all occurrences of 'r' by the right operand, M(i, 3), and all occurrences of 't' by the row number, i. The row numbers (integers) which appear in the code are to be interpreted as the names of temporary storage locations. Figure 17 is an algorithm for performing this translation. N is the number of rows in the matrix, M, C is the code output matrix, PAT1(x) a function whose value is the index of the first line of the pattern for the operator x, and PAT2(x) a function whose value is the index of the last line of the pattern for x (both these functions are defined in Figure 18). The translation of the matrix in Figure 1 is shown in Figure 19.

It is immediately obvious that very inefficient machine code is produced by this algorithm.

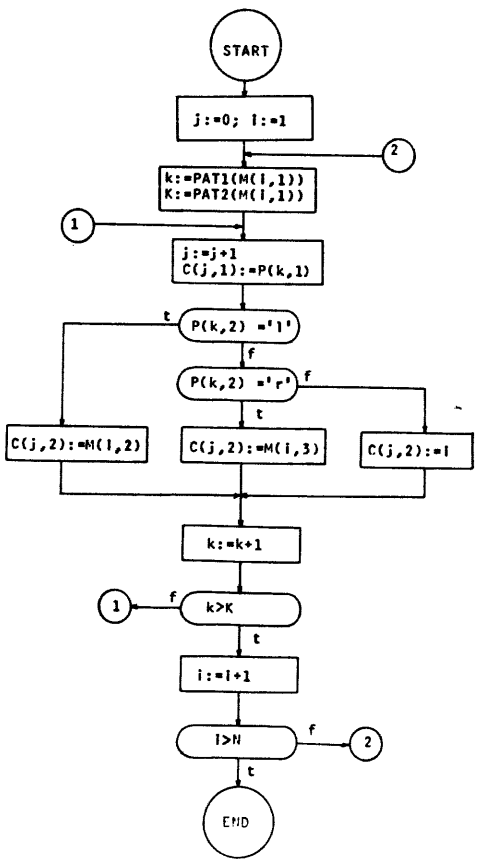


Figure 17.

x	PAT1(x)	PAT2(x)
+	1	3
-	4	6
*	7	9
/	10	12
:=	13	14

Figure 18.

1.	* B C	LDQ B FMP C STO 1
2.	- E F	CLA E FSB F STO 2
3.	* D 2	LDQ D FMP 2 STO 3
4.	+ 1 3	CLA 1 FAD 3 STO 4
5.	:= A 4	CLA 4 STO A

Matrix Machine code

Figure 19.

Once we begin to consider the production of efficient machine code, the algorithms rapidly get very complicated. We can make some improvement, however, without a great deal of complication. In the example are several redundant store and fetch instructions. These can be eliminated if we keep track of the contents of the special machine registers. We then insert store instructions only when the current contents of a special register is not one of the operands of the next operator and fetch instructions only when the operand is not already in a special register. To implement this we generalize the idea of patterns. Corresponding to each operator is a variable pattern, that is, the instructions which are actually copied into the code output matrix depend upon the contents of the special registers.

The method used in the MAD translator is general enough for the machine structure assumed in this paper. The problem of generating efficient machine code is a very difficult one and is yet unsolved. There are methods, undocumented, other than the one to be described but none which can claim to produce highly efficient code in all circumstances. The patterns will be arranged so that the result of a row is, in general, not stored, i.e., it will be left in one of the registers AC or MQ. The machine code produced when a row of the matrix is translated will depend on the values of four Boolean variables. These variables are named AC, MQ, LO, and RO. Suppose we are ready to translate the ith row of the matrix, then these variables have the following meanings:

1. If AC is true, the result produced by row i-1 is in the AC.
2. If MQ is true, the result produced by row i-1 is in the MQ.
3. If LO is true, the left operand in row i is i-1 (i.e., the result of row i-1).

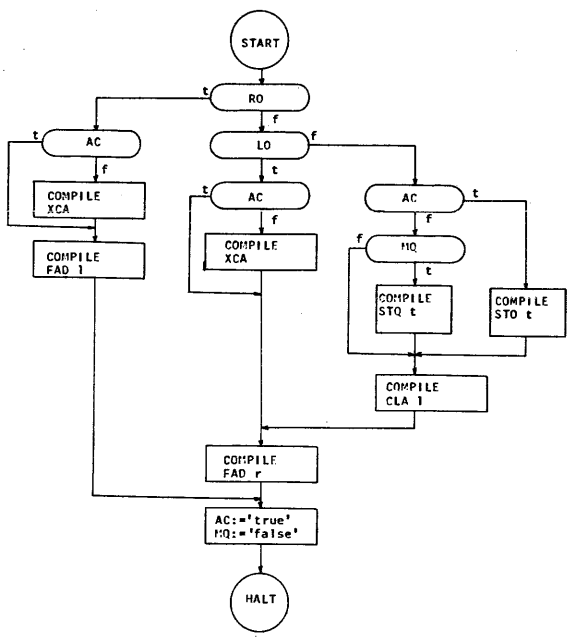


Figure 20.

4. If RO is true, the right operand in row i is i-1.

Instead of regarding the patterns as a fixed set of machine code to be produced in translating a row of the matrix, we now take the view that the pattern is really a small program which, when executed, produces machine code. Viewing it in this light, we need a language in which to write the program.

operation code	first operand	second operand	meaning
M	inst	a	compile the machine instruction inst with a in its address field (a may be l, r, t, or blank)
AC / MQ / LO / RO	l1	l2	if AC / MQ / LO / AO = 'true' transfer to line l1, otherwise transfer to line l2
S	AC / MQ	val	set the value of AC / MQ to val
J	l		transfer to line l
H			halt

In our language we will need the following types of instructions: produce a machine instruction, branch on the truth or falsity of one of the Boolean variables, absolute transfer, set the value of one of the Boolean variables, and halt. Figure 20 is a flow chart for a program to produce the code for '+', where 'l' and 'r' have the same meanings as in Figure 16, but 't' now refers to the temporary used to store the result of the previous row. Notice that if there is a result in the AC or MQ and it is not either of the operands then instructions to store it and fetch one of the operands are generated. If one of the operands is in the wrong special register an exchange instruction is generated. The word COMPILE means, here, write the instruction into the code output matrix.

A command in our pattern programming language will have a line number, an operation

x	PAT(x)
+	1
-	20
*	37
/	54
:=	68

Figure 21.

```
 1 RO  2  8        26 AC 28 27        51 MQ 52 49
 2 AC  4  3        27 M  XCA          52 M  STO t
 3 M   XCA         28 M  FSB r        53 J  49
 4 M   FAD 1       29 J  5            54 LO 55 61
 5 S   AC  true    30 AC 31 34        55 AC 57 56
 6 S   MQ  false   31 M  STO t        56 M  XCA
 7 H               32 M  CLA 1        57 M  FDP r
 8 LO  9  13       33 J  28           58 S  AC  false
 9 AC 11  10       34 MQ 35 32        59 S  MQ  true
10 M   XCA         35 M  STO t        60 H
11 M   FAD r       36 J  32           61 AC 62 65
12 J   5           37 RO 38 42        62 M  STO t
13 AC 14  17       38 MQ 40 39        63 M  CLA 1
14 M   STO t       39 M  XCA          64 J  57
15 M   CLA 1       40 M  FMP 1        65 MQ 66 63
16 J   11          41 J  5            66 M  STO t
17 MQ 18  15       42 LO 43 47        67 J  63
18 M   STO t       43 MQ 45 44        68 RO 69 74
19 J   15          44 M  XCA          69 AC 70 72
20 RO 21  25       45 M  FMP r        70 M  STO 1
21 AC 23  22       46 J  5            71 H
22 M   XCA         47 AC 48 51        72 M  STO 1
23 M   CHS         48 M  STO t        73 H
24 J   4           49 M  LDQ 1        74 M  CLA r
25 LO 26  30       50 J  45           75 J  70
```

Pattern Programs

Figure 22.

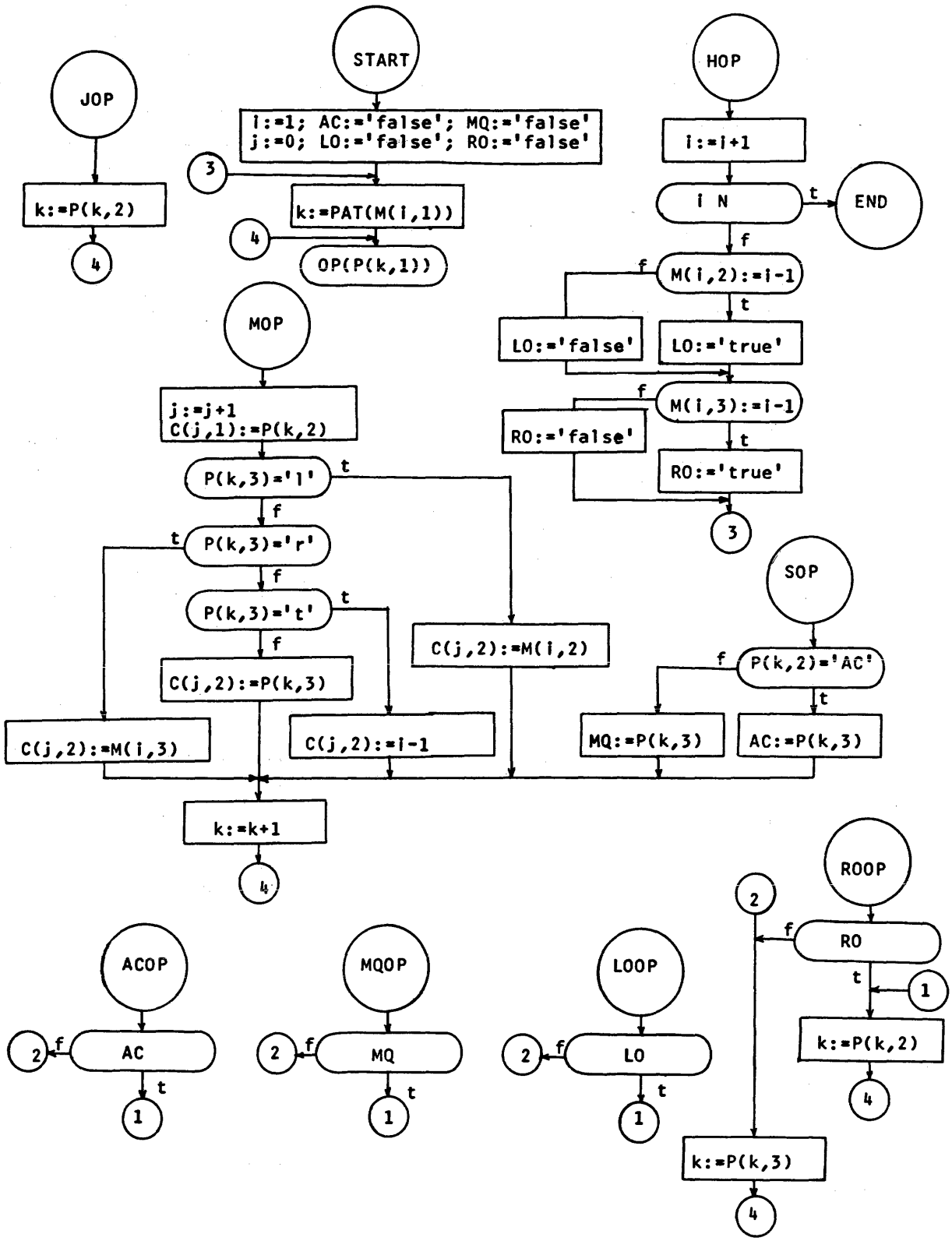

START
i:=1; AC:='false'; MQ:='false'
j:=0; LO:='false'; RO:='false'
3
k:=PAT(M(i,1))
4
OP(P(k,1))
JOP
k:=P(k,2)
4
MOP
j:=j+1
C(j,1):=P(k,2)
P(k,3)='1'
t
f
P(k,3)='r'
t
f
P(k,3)='t'
t
f
C(j,2):=P(k,3)
C(j,2):=M(i,3)
C(j,2):=i-1
C(j,2):=M(i,2)
k:=k+1
4
HOP
i:=i+1
i N
t
END
f
M(i,2):=i-1
f
t
LO:='false'
LO:='true'
M(i,3):=i-1
f
t
RO:='false'
RO:='true'
3
SOP
P(k,2)='AC'
f
t
MQ:=P(k,3)
AC:=P(k,3)
ACOP
2
f
AC
t
1
MQOP
2
f
MQ
t
1
LOOP
2
f
LO
t
1
ROOP
2
f
RO
t
1
k:=P(k,2)
4
k:=P(k,3)
4

Figure 23.

x	OP(x)
M	MOP
AC	ACOP
MQ	MQOP
LO	LOOP
RO	ROOP
S	SOP
J	JOP
H	HOP

Figure 24.

	M	AC	MQ	LO	RO	C
1.	* B C	true	false	false	false	LDQ B FMP C
2.	- E F	true	false	false	true	STO 1 CLA E FSB F
3.	* D 2	true	false	false	true	XCA FMP D
4.	+ 1 3	true	false	false	true	FAD 1
5.	:= A 4					STO A

Figure 25.

code, and two operand fields. The commands are:

The program for translating the matrix into machine code now becomes an interpreter which executes (interpretively), for each row of the matrix the appropriate pattern program. The pattern programs will be stored in a matrix, P (indexed by k), the first column is the operation code, the second column is the first operand, and the third column is the second operand. As before, we have a function PAT(x) whose value is the index of the first line of the pattern program for the operator x. The values of LO and RO are set by examining the operands in a row before executing the pattern program for that row. The function PAT(x) is defined by the table of Figure 21, Figure 22 gives the pattern programs (the P matrix), and Figure 23 is the interpreter for the pattern programs, i.e., the algorithm for translating the matrix into machine code. OP(y) is a function, defined in Figure 24, whose argument, y, is one of the operation codes of the pattern programming language and whose value is a label, the label of that portion of the algorithm which interprets the operation code y. Figure 25 shows the translation of the matrix in Figure 1 into machine code using the algorithm of Figure 23. For each row of the matrix is shown the machine code produced for that row and the status of the four Boolean variables after translating that row and just before considering the next row, that is, at point 3 in the flow chart of Figure 23. Notice that just this simple consideration of the contents of the special registers gives us a saving of five instructions when compared to the instructions produced by the algorithm of Figure 17.

It is obvious that only trivial modifications are necessary to be able to use the pattern program interpreter with Ross's intermediate form. Instead of considering the rows of the matrix in sequence, the minor and major evaluation strings are followed. When the rules for following the evaluation strings call for evaluation of a row, the appropriate pattern program is interpreted. Evans uses an algorithm very similar to the pattern program one which we have described.

No consideration has been given to the type (real, integer, etc.) of the identifiers involved. In many computers there are different instructions for floating decimal (real) and integer arithmetic. This is easily taken care of by having, for example, two pattern programs for '+', one to be interpreted when the operands are real, and the other to be interpreted when the operands are integer. Finally, it is clear that the interpreter instead of generating machine instructions could actually execute them, thus turning the entire translator itself into

an interpreter which could execute (interpretively) programs written in the source language.

BIBLIOGRAPHY

1. CHEATHAM, T. E., JR., and K. SATTLEY: Syntax Directed Compiling, *Proceedings of the 1964 SJCC,* Washington, D. C., April 19.
2. FLOYD, R. W.: Syntactic Analysis and Operator Precedence, *J. ACM,* 10 (3): 316–333 (1963).
3. FLOYD, R. W.: Bounded Context Syntactic Analysis, *Comm. ACM,* 7 (2): 62–65 (1964).
4. EVANS, A., JR.: An ALGOL 60 Compiler, paper presented at the 18th Annual Meeting of the ACM, Denver, August 1963.
5. ARDEN, B. W., and R. M. GRAHAM: On GAT and the Construction of Translators, *Comm. ACM,* 2 (7): 24–26 (1959); correction, *Comm. ACM,* 2 (11): 10–11 (1959).
6. ARDEN, B. W., B. A. GALLER, and R. M. GRAHAM: The Internal Organization of the MAD Translator, *Comm. ACM,* 4 (1): 28–31 (1961).
7. ARDEN, B. W., B. A. GALLER, and R. M. GRAHAM: An Algorithm for Translating Boolean Expressions, *J. ACM,* 9 (2): 222–239 (1962).
8. GRAHAM, R. M.: Translator Construction, *Notes of the Summer Conference on Automatic Programming,* University of Michigan, June 1963.
9. EVANS, A., JR., A. J. PERLIS, and H. VAN ZOEREN: The Use of Threaded Lists in Constructing a Combined ALGOL and Machine-Like Assembly Processor, *Comm. ACM,* 4 (1): 36–41 (1961).
10. BAUER, F. L., and K. SAMELSON: Sequential Formula Translation, *Comm. ACM,* 3 (2): 76–83 (1960).
11. FLOYD, R. W.: A Descriptive Language for Symbol Manipulation, *J. ACM,* 8 (4): 579–584 (1961).
12. ROSS, D. T., and J. E. RODRIGUEZ: Theoretical Foundations for the Computer-Aided Design System, *Proceedings of the 1963 SJCC,* Detroit, May 1963, pp. 305–322.
13. NAUER, P., et al.: Report on the Algorithmic Language ALGOL 60, *Comm. ACM,* 3 (5): 299–314 (1960).

APPENDIX A

Definition of the Language Used in the Examples

<identifier>::=A | B | C | D | E | F | G | H

<primary>::=<identifier> |
(<expression>)

<mop>::=* | /

<aop>::=+ | —

<term>::=<primary> | <term><mop>
<primary>

<expression>::=<term> | <expression>
<aop> <term>

<assignment statement>::=<identifier>:=
<expression>

APPENDIX B

The Machine Language Used in the Examples

There are two special registers, the AC and the MQ. Instructions are single address. The meaning of the instructions is expressed as a short ALGOL program. The '?' in the context 'MQ:=?' means that the contents of the MQ is indeterminate. When the name of a register does not appear to the left of an ':=' in the description of an instruction, then the value of that register is unchanged.

Instruction		Meaning
CLA	X	AC:=X
FAD	X	AC:=AC+X; MQ:=?
LDQ	X	MQ:=X
FMP	X	AC:=MQ*X; MQ:=?
FDP	X	MQ:=AC/X; AC:=?
FSB	X	AC:=AC—X; MQ:=?
STO	X	X:=AC
STQ	X	X:=MQ
CHS		AC:=—AC
XCA		TMP:=AC; AC:=MQ; MQ:= TMP

SYNTAX-DIRECTED COMPILING

T. E. Cheatham, Jr., and Kirk Sattley
Computer Associates, Inc.
Lakeside Office Park
Wakefield, Massachusetts

INTRODUCTION

This paper is primarily concerned with the analysis of source statements in a programming language, although some of the ideas and techniques may be applicable to the analysis of source statements in a natural language. We are particularly concerned with those techniques which might be classed as predictive; the companion paper by Graham[7] is concerned with other ("nonpredictive") techniques of analysis. Very broadly the techniques we will discuss operate as follows: Given a set of rules (Syntax Specification) for forming allowable constructs, eventually resulting in a *statement* (or sentence, word, program, etc.) of a language, we analyze a source statement in that language by guessing, or predicting, how the statement is constructed and either verifying that this is the case or backing up to try again, assuming some other method of construction. We keep a "history" of our attempts and when we have determined the exact way in which the statement is constructed we can use this "history" of its construction for further processing of the components of the statement.

We will be concerned, secondarily, with the synthesis of machine coding, given an analysis of a source statement. We do not, however, discuss in any detail the difficult (and, at this point, not terribly well understood) problems of synthesizing highly efficient coding. Reference [1] contains a brief discussion of this problem.

We are concerned hardly at all with the extremely important and often neglected problems of the environment in which a compiler or code resulting from a compiler is to operate. Reference [3] sketches our position in this matter.

The phrase "syntax-directed" in the title refers to the method by which the compiler is given the syntactic specification of the language it is to compile. That is, rather than having the syntactic structure of the language reflected in the actual encoding of the compiler algorithm, a "syntax-directed" compiler contains (or uses, as parametric data) a relatively simple and direct encoding of the syntactic structure of the language, for example, as it might be expressed in Backus Normal Form. By "simple and direct encoding," we mean, for instance, numerical codes for the distinct syntactic types of the language, and direct pointers representing the relations of concatenation and alternative choice, plus perhaps some sorting.

This paper is not intended as a review or critique of syntax-directed compilers or compiler techniques nor have we presented a comprehensive bibliography on the subject. Rather, our purpose is tutorial—to present in as straightforward a manner as possible the essential ideas of syntax-directed compilers. Unfortunately, there is, at the present time, no completely adequate review paper on the subject; Floyd[13] does include a rather complete bibliography.

Our presentation commences with a discussion of syntax and the syntactic specifications of languages—programming languages in particular. We then discuss techniques for encoding the syntax into tables and develop a simple algorithm, the ANALYZER, which can perform a syntactic analysis of source material, using this tabled syntax specification as data. From this we proceed to a discussion of the generation or synthesis of code from the results of the analysis. Finally, we discuss several problems and limitations. Certain problems of syntactic specification and some modifications of the schemes we describe in the body of the paper have been discussed in an appendix.

SPECIFICATION OF SYNTAX

Several essentially equivalent formalisms for the representation of syntax have been developed. These include such things as

Post Production Systems, developed by the logician Emil Post during the 1940's as a tool in the study of Symbolic Logic;

Phrase Structure Grammars, developed by the linguist Noam Chomsky during the 1950's as a tool in the study of natural languages; and

Backus Normal Form, developed by the programmer John Backus during the late 1950's as a tool in the description of programming languages.

We shall use here a formalism most similar to Backus's.

A syntactic specification of a language is a concise and compact representation of the structure of that language, but it is merely that—a description of structure—and does not by itself constitute a set of rules either for producing allowable strings in the language, or for recognizing whether or not a proffered string is, in fact, an allowable string.

However, rules can be formulated to produce, or recognize, strings according to the specification. In a "syntax-directed" compiler it is an *algorithm* which performs the recognition of allowable input strings, and it does this by using (an encodement of) the Syntax Specification as data. In this paper, we shall call such an algorithm an (or the) "Analyzer."

In order to discuss the structure of the language, we give names to classes of strings in the language—we call these names (or the classes they denote) "Syntactic Types." Some of the classes of interest consist of single characters of the source alphabet: these we call "Terminal Types," and specifically "Terminal Characters"; to talk about any particular one, we will merely display the character. Most of the classes, though, are more complicated in structure and are defined in terms of other classes; these we call "Defined Types," and to designate one, we choose a mnemonic name for the class and enclose it in the signs '<' and '>'.

Basic Syntax Specification

Rather than proceed immediately to a discussion of Backus Normal Form, we shall first define a simple form of Syntax Specification—the Basic Specification. This consists of a set of "Simple Type Definitions" (meaning, not that the Types are simple, but the Definitions are). A Simple Type Definition consists of the name of a Defined Type, followed by the curious sign '::=' followed by a sequence of Syntactic Types, Defined or Terminal. An example, taken from the Syntax I—soon to be discussed—would be:

<assignment>::=<variable>=<arithexpr>

The Defined Type on the left of the '::=' is called *The* Defined Type of the Definition; and the Definition is said to be *a* Definition of its defined type. In general—even for the more complicated forms of Type Definitions yet to come—we shall call the right-hand side of the Definition the "Definiens." Any sequence of type designators appearing in a Definiens is called a "Construction," and each type designator within the Construction is a "Component" of the Construction. So, the above example is a Definition of the Defined Type <assignment>; its Definiens is a Construction with three components, which are, in the order of their appearance, the Defined Type <variable>, the Terminal Charcater '=' and the Defined Type <arith expr>.

A Simple Type Definition of this sort states that, among the strings of the source language belonging to the Defined Type, are those which are concatenations of substrings—as many

substrings as there are components of its (Simple) Definiens—such that each substring (in order of concatenation) belongs to the Syntactic Type named by the corresponding Component (in order of appearance in the Definiens). Applied to our example: A source string belongs to the class <assignment> (or, for short, "is an <assignment>") if it can be broken into three consecutive substrings, the first of which is a <variable>, the second of which is the single character '=', and the third of which is an <arith expr>.

If we were interested in using Syntax Specifications as "'generative grammars"—that is, if we were writing an algorithm to use a Syntax Specification to produce samples of strings of the language, we would write something which, applied to our example, would have the effect of: "if you wish to produce an <assignment>, then: *first* choose any definition of <variable> and produce a string according to that definition, then, *second* write down the character '=', then *third* produce an <arith expr> according to any definition of that type; then you have produced an <assignment>"

Thus, the use of a (Basic) Syntax Specification as a generative grammar is quite straightforward. The inverse problem—using the Syntax Specification as a "recognition grammar"—is, like many inverse problems, rather more involved. In our opinion, the fundamental idea —perhaps "germinal" would be a better word —which makes syntax-directed analysis by computer possible is that of *goals*: a Syntactic Type is construed as a goal for the Analyzer to achieve, and the Definiens of a Defined Type is construed as a recipe for achieving the goal of the Type it defines.* It is this use of goals which leads to another description of analysis techniques of this kind—"predictive analysis": setting up the recognition of a particular Syntactic Type as a goal amounts to predicting that an instance of that type will be found. Needless to say, this use of the term "goal" is not to be confused with the "goal-seeking behavior" of "artificial intelligence" programs or "self-organizing systems." However, when we come to specifying the algorithm for selecting particular goals in a particular order, we reach the point at which the several existing syntax-directed techniques diverge. Our purpose in this section on "Basic Syntax Specification" is to lay a foundation common to the principal different applications of the technique; hence, if we try to "picture" the use of a Syntax Specification as a recognition grammar, as we pictured its use as a generation grammar in the preceding paragraph, the most generally valid statement we can make is:

We can say that we have recognized an occurrence of a given Syntactic Type (at a given position in the source string) if one of the two following conditions obtains:

1. The Syntactic Type is a Terminal Character, and the character at the given position in the source string is exactly that Terminal Character;
2. The Syntactic Type is a Defined Type, and for some one of its (Simple) Definitions, we *have already recognized* concatenated occurrences of the Components of that Definiens, in the stated order, the first of which occurs at the given position.

In order for the set of Simple Type Definitions to constitute a useful Syntax Specification, it should satisfy some conditions.

(C1) Any Defined Type which occurs as a Component in any Definiens must also occur as the Defined Type of some definition.

The desirability of this "completeness condition" is fairly obvious—it will be very difficult to recognize a Defined Type if the Analyzer has no Definition of that Type. Of course, it is possible that the Analyzer may never be asked to find an instance of this Type, but then all the Definitions which included it as a Component would also be superfluous.

(C2) Every Defined Type must ultimately be constructible entirely out of Terminal Characters.

This "connectedness condition" is designed to prevent a cycle of definitions which it is impossible to break out of—that is, if a Defined Type is defined *only* in terms of Defined Types,

* To our knowledge, the first person to formulate and implement this conception was E. T. Irons, in his initial design of the PSYCO compiler; his paper [4] describes the essentials of his compiling technique.

each of which in turn is defined only in terms of Defined Types, etc. Of course, it will be true that there will be cycles within the act of definitions, and these cycles may be traversed arbitrarily many times; but there must exist some point in each cycle where an alternative definition of one of the types exists. It is probably sufficient to restate condition (C2) in the following fashion:

A Terminal Type will be said to be "grounded." A Defined Type is grounded if it has at least one Definition, *all* of whose Components are grounded; then

(C2′) Every Defined Type must be grounded.

(C3) There must exist exactly one Defined Type which does not appear as a Component in any Definiens (except possibly its own). This Type is called the "Starting Type" of the Syntax Specification.

The Starting Type represents the "largest" construction allowable under the Specification —e.g., "sentence," or perhaps "paragraph," in natural language applications, or usually "program" in compiler applications. If there is no Starting Type, the Analyzer, quite literally, will not known where to begin.

Let us note here in passing that there is a property of Syntax Specifications which is of great importance to theoreticians in this field, and to people who are designing new languages or trying to construct Specifications for existing complex languages, but which is irrelevant to the problem of programming a syntax-directed Analyzer. This is the question of "structural ambiguity"—does the Syntax Specification permit a particular source-language string to be *correctly* analyzed in two different fashions? A simple example, taken from natural language (with apologies to Oettinger) is: "Time flies incessantly." This is certainly an English sentence—but is it a metaphorical declarative sentence, or a terse imperative? In the case of an Analyzer algorithm on a computer, only one thing is done at a time—if the Analyzer is asked to find an instance of an Ambiguous Syntactic Type, it must try one of the possible definitions first; if that definition succeeds, the Analyzer is satisfied, and the other definitions are not considered. This is not to say that an Analyzer, one of whose functions is to find all possible analyses, cannot be built; this has been done by Oettinger[11] for natural language, and by Irons[5], for use in compiling.

Some Transformations of the Basic Specification

We shall now proceed to build up to the description of a particular simple Analyzer algorithm, and at this point, we must choose one among several different techniques. The differences between the various techniques stem from the following considerations:

—Given a Syntax Specification, there are different ways of using it to determine the next goal which the analyzer is to pursue. The two major approaches are called the "top-down" and the "bottom-up" techniques.

—There are different ways to use the output of the Analyzer, e.g., interpretation, immediate generation of output code, recording of the analyzed structure for later generation, etc.

The particular type of Analyzer we have chosen to describe here is, we believe, the easiest to explain, and is suitable for any of the three output-treatments mentioned above. It does not correspond, so far as we know, to any actually existing compiler system, although it bears a surprisingly strong resemblance to the algorithm used in some of the compilers that Computer Associates, Inc., has recently produced. (See Shapiro and Warshall)[1].

The first step is to transform our Basic Syntax Specification into a Backus Normal Form. The description of a Syntactic Type Definition is now expanded so that the Definiens, instead of simply a Construction (which, remember, was a sequence of Components, which, in turn were Syntactic Types) can now be a sequence of Constructions, separated by the special sign '|'. Any such sequence of Constructions, separated by '|' and appearing in a Definiens is called an "Alternation," and the individual Constructions in the sequence are called "Alternatives" of the Alternation. To transform a Basic Syntax Specification into Backus Normal Form, we must repeatedly apply the following transformation rule to the set of Definitions, until it can no longer be applied:

(T1) If any Defined Type has more than one Definition in the set, delete all such Definitions, and add to the set a new Definition whose left-hand side is the Defined Type in question, and whose Definiens is an Alternation of the original (Basic) Definientes.

As an example, the Basic Syntax Specification for the simple language we are using for illustration in this paper would have contained three definitions for <factor>:

<factor> ::= <variable>
<factor> ::= <integer>
<factor> ::= (<arith expr>)

After applying (T1) to the Basic Specification, these three Definitions would be replaced by the single Definition

<factor> ::= <variable> | <integer> | (<arith expr>)

This Definition, of course, should be read "a source string is a <factor> if it is *either* a <variable> *or* an <integer> *or* an <arith expr> enclosed in parentheses." This Backus Normal Form is exactly the form of Syntax Specification used in the defining documents for ALGOL 60 [8], and Table 1 presents a complete syntax for a simple arithmetic programming language in this form, which we shall refer to as "Syntax I."

The Action of the Analyzer

We can now sketch out the action of the Analyzer: At the beginning of the process, it takes the Starting Type of the Specification as its first goal. Then at any point in the process it follows these steps when it has a Defined Type as its current goal:

The Analyzer consults the Definition of the Defined Type (in Backus Normal Form, of course, each Defined Type has a unique Definition), and specifically, it considers the first Alternative in that Definition. It then successively takes each Component of that Alternative as a sub-goal. (Of course, it must re-enter itself for each of these goals, and it must keep track of where it was at each level of re-entry.) If at any point it fails to find one of these sub-goals, it abandons that Alternative, and considers the next Alternative in that Definition, if there is one, and steps through the Components of that Alternative. If there is no next Alternative, it has failed to realize its current goal, and reports this fact "upstairs." If it succeeds in finding the sub-goals corresponding to each of the Components of any Alternative in the Definition of its current goal, it has found its goal, and reports that fact.

This rough sketch conveniently ignores a number of sticky points which we now have to consider. The first of these points is that we discussed the action of the Analyzer only when its current goal was a Defined Type. What if the goal is a Terminal Character?

When it comes to writing a compiler in practice, the question of recognizing Terminal Characters brings us face to face with the lovely problems of restricted character sets, input-output idiosyncracies of the particular computer, etc. Both in practice and in the remainder of this paper, we assume the presence of another routine, called the "Recognizer," which the Analyzer can call upon to deal with these problems. So far, we have also glossed over the problem of keeping track of where in the Input String the Analyzer is looking. Obviously, when the first Component of some Construction has been recognized, starting at a certain point in the Input String, then, when the Analyzer proceeds to look for the next Component, it must move its Input-String pointer past the substring which satisfied the first Component. Now, since a Type which has been successfully recognized consists, ultimately, of a sequence of Terminal Characters, and the recognition of Terminal Characters is the job of the Recognizer, we shall also leave the moving of the Input-String pointer to the Recognizer. The fundamental action of the Recognizer is then as follows:

The Recognizer is called by the Analyzer, and asked if a specified Terminal Character occurs at a stated character position in the Input String. The Recognizer then negotiates with the I/O system of the computer (if necessary) and examines the character-position in the input string. If the input character at that position is *not* the Terminal Character the Analyzer asked for, the Recognizer reports failure. However, if the input character *is* the desired Terminal Character,

the Recognizer reports success to the Analyzer, *and* advances the Input-string pointer by one character position.

Having the Recognizer at hand, it turns out to be convenient in practice to give it some further responsibilities. Consider the definitions of <variable> and <integer> in Syntax I. These amount to saying that an <integer> is any sequence of digits, and a <variable> is any sequence of letters or digits as long as the first one is a letter. If we relegate the recognition of these fundamental types to the Recognizer, rather than the Analyzer, we obtain several advantages.

—The Recognizer can be hand-tailored to perform these particular recognitions very efficiently on the particular machine, and this speeds up the analysis considerably.

—As far as the Analyzer is concerned, if the Syntax Specification calls for an <integer>, for instance, any old integer will do. But when we come to generating output code, we'll need to know the particular integer which occurred at that point. The Recognizer can perform the conversion from external number representation to machine representation, and either return the numerical value, or enter the number in a "Literal Table" and return its index value. Similarly, when it recognizes <variable>, it can look in a "Symbol Table" for previous occurrences of that particular variable, add it to the table if necessary, and return a line number.

—In practical applications the question of what constitutes a "blank" is often an involved one. In some languages, a long comment may function syntactically as a blank. When a compiler runs under the control of some operating systems, certain segments of the Input string (e.g., identification fields in cards) must be treated as blanks, or ignored entirely. Since the Recognizer constitutes the interface between the Analyzer and the outside world, it can take care of these matters.

To allow for this extended Recognizer in our Syntax Specification, we allow another sort of Terminal Type (up to now, we recall, the only Terminal Types have been Terminal Characters). We designate these new Terminal Types with script capital letters, and call them "Terminal Classes." Thus, in Syntax I, we can delete the definitions of <variable>, <integer>, <letter>, and <digit>, and replace the Definientes of <variable> and <integer> by the Terminal Classes $\mathcal{V}$ and $\mathcal{J}$, respectively. This produces Syntax II, Table 1, which is the one we shall refer to throughout the rest of this paper.

But this technique could be carried further. A compiler-builder might decide that he prefers operator-precedence techniques for the analysis of arithmetic expressions, while keeping the flexibility of syntax-direction for analysis of the larger constructions. His arithmetic-expression scanner would then function as a Recognizer for the previous Defined Type '<arith expr>,' and, for this simple language, the Syntactic Specification would take the form of Syntax III, Table 1.

To summarize: When the current goal is a Defined Type, the Analyzer calls upon itself to find it, but when the goal is a Terminal Type, it calls upon the Recognizer. When the Recognizer is called, it determines according to its own internal rules, whether the desired Terminal Type occurs in the Input string at the current pointer-position; if not, it reports failure; if so, it advances the pointer past the substring which constitutes the Terminal Type (single character, or member of a Terminal Class), and reports success.

Encoding the Syntax Specification

We are now almost ready to proceed to the encoding of the Syntax Specification for the use of the Analyzer, except for one embarrassing question:

Consider, as an example, the definition of <term> in Syntax II:

<term> ::= <factor> | <term> * <factor>

What if the Analyzer should find itself considering the second Alternative in this Definition? This would amount to the Analyzer saying to itself "in order to find my current goal, which is <term>, I must set out to find the first Component of this Alternative, which is <term>." In order to find a term it must

TABLE 1
Alternative Syntax Specifications

Syntax I:

```
<program>     ::= <assignment> | <assignment> ; <program>
<assignment>  ::= <variable> = <arith expr>
<arith expr>  ::= <term> | <arith expr> + <term>
<term>        ::= <factor> | <term> * <factor>
<factor>      ::= <variable> | <integer> | (<arith expr>)
<variable>    ::= <letter> | <variable> <letter> | <variable> <digit>
<integer>     ::= <digit> | <integer> <digit>
<letter>      ::= A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P |
                  Q | R | S | T | U | V | W | X | Y | Z
<digit>       ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

Syntax II:

```
<program>     ::= <assignment> | <assignment> ; <program>
<assignment>  ::= <variable> = <arith expr>
<arith expr>  ::= <term> | <arith expr> + <term>
<term>        ::= <factor> | <term> * <factor>
<factor>      ::= <variable> | <integer> | (<arith expr>)
<variable>    ::= 𝒱
<integer>     ::= 𝒿
```

Syntax III:

```
<program>     ::= <assignment> | <assignment> ; <program>
<assignment>  ::= <variable> = <arith expr>
<arith expr>  ::= ℰ
<variable>    ::= 𝒿
```

be able to find a term first. This is called the "left recursion problem," and it has led some language designers to disallow Type Definitions which include Alternatives which mention the Defined Type of the Definition as their first Component. To a human analyst, of course, the intent of the Definition is plain; he should first look for a <factor>; if he finds one, he has indeed found a <term>, but he should continue looking to see if he can find a '*' followed by another <factor>; if he can, he has found a "longer" <term>, and should continue looking for a still longer one; as soon as he fails to find a '*' following his latest <term>, he can stop looking, confident that he has found the longest <term> at that point in the string. This recognition process can be embodied in the encoding of the Syntax Specification, but it does require detecting the presence of these left-recursive alternatives, and giving them some special treatment. Keeping this in mind, we shall proceed to encode the Syntax Specification.

The encoding consists of two tables, the Syntax Type Table and the Syntax Structure Table. The Syntax Type Table will contain an entry for each Syntactic Type which occurs anywhere in the Syntax Specification, whether it be a Defined Type or a Terminal Type. Each entry i the Type Table consists of two items: a yes-no item TERMINAL, and an integer item LOOKFOR. When line t in the Type Table corresponds to a Terminal Type, TERMINAL [t] will be set to "yes," and LOOKFOR [t] will contain an arbitrary code number which the Recognizer will interpret as denoting the particular Terminal Character or Terminal Class it should try to recognize. When line t in the Type Table

TABLE 2
The Syntax Tables

(GOAL) (Type)	(Index)	TERMINAL	LOOK-FOR
<program>	i	No	1
<assignment>	ii	No	4
<arith expr>	iii	No	7
<term>	iv	No	10
<factor>	v	No	13
<variable>	vi	No	18
<integer>	vii	No	19
$\mathcal{V}$	viii	Yes	101
$\mathcal{I}$	ix	Yes	102
;	x	Yes	1
=	xi	Yes	2
+	xii	Yes	3
*	xiii	Yes	4
(	xiv	Yes	5
)	xv	Yes	6

2.1
Syntax Type Table

SOURCE (Index)	TYPE-CODE	STRUCT	SUC-CESSOR	ALTER-NATE	Corresponds to Definition
1	ii	Yes	2	FAIL	1.1
2	x	No	3	OK	
3	i	Yes	OK	FAIL	1.2
4	vi	No	5	FAIL	
5	xi	No	6	FAIL	
6	iii	Yes	OK	FAIL	1.2
7	iv	Yes	8	FAIL	3.1
8	xii	No	9	OK	
9	iv	Yes	8	FAIL	3.2
10	v	Yes	11	FAIL	4.1
11	xiii	No	12	OK	
12	v	Yes	11	FAIL	4.2
13	vi	Yes	OK	14	5.1
14	vii	Yes	OK	15	5.2
15	xiv	No	16	FAIL	
16	iii	No	17	FAIL	
17	xv	Yes	OK	FAIL	5.3
18	viii	Yes	OK	FAIL	6.1
19	ix	Yes	OK	FAIL	7.1

corresponds to a Defined Type, TERMINAL [t] will be set to "No," and LOOKFOR [t] will contain some line-number in the Syntax Structure Table, to be filled in presently. We keep in mind that we can now use small integers as, in effect, names of Syntactic Types, by using them to index the Syntax Type Table.

The Syntax Structure Table will contain a line for each Component of each Alternative of each Definiens in the Syntax Specification. Each line of the Structure Table will consist of four items:

TYPECODE, an integer item, will contain line numbers referring to the Type Table;

STRUCT, a yes-no item;

SUCCESSOR and

ALTERNATE, integer items, will contain line numbers referring to other lines of the Structure Table, plus two special codes denoted by "OK" and "FAIL."

The Syntax Structure Table is constructed according to the following rules:

Consider first a Defined Type which has no left-recursive Alternative in its Definiens. Reserve a block of entries in the Structure Table. Assign an entry in the Structure Table to each Component in each Alternative in the Definiens. In the Type Table line corresponding to this Defined Type—say, t—set LOOKFOR [t] to the Structure-Table line number assigned to the first Component of the First Alternative of the Definiens. In each Component-line s, set TYPECODE [s] to the Type Table line number of the Syntactic Type which occurs as that Component in the Definiens. In each line corresponding to a Component which is *not* the last Component of an Alternative, set STRUCT to "No" and SUCCESSOR to the line corresponding to the next Component. In each line corresponding to a Component which *is* the last Component of an Alternative, set STRUCT to "Yes" and SUCCESSOR to "OK." In each line corresponding to a first Component of any Alternative except the last Alternative of the Definiens, set ALTERNATE to the line corresponding to the first component of the next Alternative. Set all other ALTERNATE fields to "FAIL."

If the Defined Type contains a left-recursive Alternative: (we shall here assume there is only one left-recursive Alternative—See Appendix). Set the left-recursive Alternative aside temporarily, and carry out the above process for the other Alternatives. Then:

Assign a Structure-Table line to each Component of the recursive Alternative *except* the recursive Component itself.

Set TYPECODE in each of these lines, as above.

Set SUCCESSOR and STRUCT in each of these lines, except for the last Component, as above.

Call the first of these lines (the one corresponding to the Component which immediately follows the recursive Component in the Definiens) the "handle."

Set ALTERNATE in each of these lines, except the handle, to "FAIL."

Set ALTERNATE in the handle to "OK."

Set SUCCESSOR in the line for the last Component of this Alternative to the handle, and set STRUCT in this line to "Yes."

Now, in the lines corresponding to last Components in all the other Alternatives in this Definiens, SUCCESSOR will have been set to "OK" by the nonrecursive treatment. *Replace* each of these "OK"s by the line number of the handle.

The Syntax Type Table and the Syntax Structure Table corresponding to Syntax II are shown together as Table 2. In the hope of

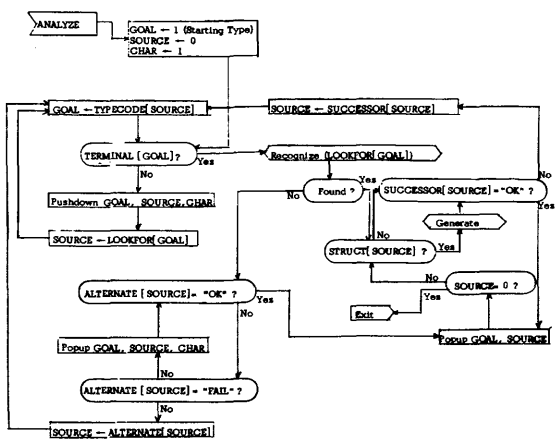


Figure 1. Flow Chart for ANALYZE.

reducing the confusion involved in having entries in each table pointing to the other, we have indicated the indexing of the Syntax Type Table with Roman numerals, and the indexing of the Syntax Structure Table with Arabic numerals, and we have added the names of the syntactic types corresponding to the lines of the Syntax Type Table.

The Analyzer Algorithm

The flow chart, Figure 1, illustrates an Analyzer working from Syntax Tables of the sort we have just constructed. The following remarks will help to follow the flow chart.

—The global quantities GOAL, SOURCE, and CHAR are used as follows:

GOAL is the line number in the Syntax Type Table corresponding to the Syntactic Type currently being considered (initially, the Starting Type).

SOURCE is the line number in the Syntax Structure Table of the Component currently being considered (initially undefined, hence set to zero).

CHAR is the (ordinal) number of the character position in the Input String next to be considered (initially set to the beginning of the String, here 1).

—The operations of "Pushdown" and "Popup" are performed on these globals—this may be done by any appropriate mechanism. For definiteness, let us assume an index Y (initially zero) accessible to these two operations, and arrays GYOYO, SYOYO, and CYOYO. Then (in quasi ALGOL notation),

```
Pushdown: Y := Y + 1
          GYOYO[Y] := GOAL;
          SYOYO[Y] := SOURCE;
          CYOYO[Y] := CHAR;

Popup:    GOAL := GYOYO[Y];
          SOURCE := SYOYO[Y];
          if CHAR is mentioned in the call
          then CHAR := CYOYO[Y]; Y
          := Y — 1;
```

Plausibly, CHAR is popped up when an Alternative has failed, and the Analyzer must back up to the beginning of that Construction and try another Alternative at the same place; and CHAR is not popped up—hence left as it has been set by the successful recognitions—when some Alternative has succeeded.

—Recognize is assumed as described earlier: It returns a success/failure indicator which is tested in the "Found?" box. For definiteness again, we shall assume that, when it succeeds in recognizing a Terminal Class, it places a Symbol-Table or Literal-Table line number in some global location, for the Generator to use.

—The following sections of this paper will discuss possible uses to be made of the Analyzer's results. The routine which considers these results is named the "Generator," and it is represented in this flow chart by a subroutine call box: "Generate." When Generate is called, the value of SOURCE uniquely indicates the Syntactic Type which has been recognized and, moreover, the particular Alternative in the Definition of that Syntactic Type which has just succeeded. The column headed "Corresponds to Definitions" has been added to the Syntax Structure Table to indicate this relationship. The numbers in this column correspond to the Alternatives in the "semantics" tables, Tables 4 and 5.

DOING SOMETHING USEFUL WITH THE ANALYSIS

A syntactic analysis, such as that depicted verbally in the preceding section or via the flow chart in Figure 1, is an important part of the problem which must be solved by a compiler, but it is only a part. The goal of a compiler is, after all, to produce the coding required to carry out the procedure described in the programming language being compiled. This coding might be desired as actual machine instructions for some computer or it might be desired as instructions appropriate for some interpreter available on one or more machines or it might be desired in some other form. In any event, some further processing is required once the syntactic analysis is complete in order to "generate" and format the coding to be output.

Let us suppose that the syntactic analysis has proceeded to the point where some syntactic type has been recognized (in the flow chart, Figure 1, we have passed through the "GENERATE" box). The contents of SOURCE tells us

TABLE 3
"Interpretive Semantics" for Syntax II

1.1 <program>	::=	<assignment>	{Execute the <assignment> then halt}
1.2		| <assignment> ; <program>	{Execute the <assignment> then proceed}
2.1 <assignment>	::=	<variable> = <arith expr>	{"Locate" the <variable> (determine its address for later assignment of value); then evaluate the <arith expr>; then assign its value to the <variable>}
3.1 <arith expr>	::=	<term>	{Evaluate the <term>; the value of the <arith expr> is this value}
3.2		| <arith expr> + <term>	{Evaluate the <term> and then the <arith expr>; the value of the (defined) <arith expr> is the sum of these two values}
4.1 <term>	::=	<factor>	{Evaluate the <factor>; the value of the <term> is this value}
4.2		| <term> * <factor>	{Evaluate the <term> and then the <factor>; the value of the (defined) <term> is the product of these two values}
5.1 <factor>	::=	<variable>	{The value of the <factor> is the value of the <variable>}
5.2		| <integer>	{The value of the <factor> is the value of the <integer>}
5.3		| (<arith expr>)	{Evaluate the <arith expr>; the value of the <factor> is the value of the <arith expr>}
6.1 <variable>	::=	$\mathcal{V}$	{The value of the <variable> is the value most recently assigned to the variable $\mathcal{V}$}
7.1 <integer>	::=	$\mathcal{I}$	{The value of the <integer> is the value of the integer $\mathcal{I}$ (according to the conventional representation of integers)}

which syntactic type has been recognized as well as which alternative construction of the syntactic type was built. Thus, some action or set of actions could be initiated at this point to process this syntactic type in a variety of ways.

For example, Table 3 gives for each alternative construction of the syntactic types of Syntax II a verbal description of the computations to be performed upon recognition of that construction. Corresponding to this table, the analysis of the assignment statement

$$X = NU^* (Y + 15)$$

could yield the following fragments of (slightly edited) verbal description:

Line	*Construction*			*Source*	*Description of Computation*
6.1	<variable>	::=	$\mathcal{V}$	X	The value* of the <variable> is the value most recently assigned to the variable X.
6.1	<variable>	::=	$\mathcal{V}$	NU	The value of the <variable> is the value most recently assigned to the variable NU.
5.1	<factor>	::=	<variable>	NU	The value of the <factor> is the value of NU.
4.1	<term>	::=	<factor>	NU	The value of the <term> is the value of NU.
6.1	<variable>	::=	$\mathcal{V}$	Y	The value of the <variable> is the value most recently assigned to the variable Y.
5.1	<factor>	::=	<variable>	Y	The value of the <factor> is the value of Y.
4.1	<term>	::=	<factor>	Y	The value of the <term> is the value of Y.
3.1	<arith expr>	::=	<term>	Y	The value of the <arith expr> is the value of Y.
7.1	<integer>	::=	$\mathcal{J}$	15	The value of the <integer> is 15.
5.1	<factor>	::=	<integer>	15	The value of the <factor> is 15.
4.1	<term>	::=	<factor>	15	The value of the <term> is 15.
3.1	<arith expr>	::=	<arith expr> +<term>	Y + 15	The value of the <arith expr> is Y + 15.
5.3	<factor>	::=	(<arith expr>)	(Y + 15)	The value of the <factor>, (Y + 15) is Y + 15.
4.1	<term>	::=	<term> *<primary>	NU*(Y + 15)	The value of the <term>, NU*(Y+15) is NU*(Y+15).
3.1	<arith expr>	::=	<term>	NU*(Y + 15)	The value of the <arith expr> NU*(Y+15) is NU*(Y+15).
2.1	<assignment>	::=	<variable>= <arith expr> X=	NU*(Y + 15)	Assign the value of NU*(Y + 15) to the variable X.

* Obviously, the current value of the variable X is not of interest here since the purpose of the assignment is to assign a value to X.

Thus, with a certain amount of editing, the recognition of X = NU*(Y + 15) yields the verbal description:

> "Let NU and Y represent the values most recently assigned to the variables NU and Y; then compute NU*(Y + 15) and assign the resulting value to the variable X."

Table 4 illustrates a very simple approach to the problem of machine code synthesis. With each syntactic construction is associated a set of actions. These actions are of two types—*output* and *set*. The interpretation of the actions is reasonably obvious. The bracketed numerals under the components of a construc-

TABLE 4

Machine Code Semantics for Syntax II—Direct Generation

1.1 <program>	::=	<assignment> [1]	1. Output END
1.2		| <assignment> ; <program> [1] [2] [3]	----------
2.1 <assignment>	::=	<variable> = <arith expr> [1] [2] [3]	1. Output CLA (addr) [3] 2. Output STO addr [1]
3.1 <arith expr>	::=	<term> [1]	1. Set addr = addr [ ] [1]
3.2		| <arith expr> + <term> [1] [2] [3]	1. Output CLA (addr) [1] 2. Output ADD (addr) [3] 3. Output STO (addr) [ ]
4.1 <term>	::=	<factor> [1]	1. Set addr = addr [ ] [1]
4.2		| <term> * <factor> [1] [2] [3]	1. Output LDQ (addr) [1] 2. Output MPY (addr) [3] 3. Output STQ (addr) [ ]
5.1 <factor>	::=	<variable> [1]	1. Set addr = addr [ ] [1]
5.2		| <integer> [1]	1. Set addr = addr [ ] [1]
5.3		| (<arith expr>) [1] [2] [3]	1. Set addr = addr [ ] [2]
6.1 <variable>	::=	$\mathcal{V}$	1. Set addr = the variable name [ ] recognized at this point of the input string.
7.1 <integer>	::=	$\mathcal{I}$	1. Set addr = a symbolic name for the [ ] address in which will be kept the integer constant recognized at this point in the input string.

tion identify the components. The notation addr[1] means the result address associated with the component identified by [1] in the syntax specification. A bracketed blank represents the syntactic type being defined and addr[ ] represents the result address of this type; when the addr[ ] appears in a machine instruction a temporary storage register is to be assigned. In the example below we use the notation t_j for the j^{th} such temporary. Again consider the assignment

$$X = NU*(Y + 15).$$

If this assignment is analyzed and actions carried out as per Table 4, the following results:

Line	*Construct*			*Source*	*Result*
6.1	<variable>	::=	$\mathcal{V}$	X	$addr_{[<variable>]} = X$
6.1	<variable>	::=	$\mathcal{V}$	NU	$addr_{[<variable>]} = NU$
5.1	<factor>	::=	<variable>	NU	$addr_{[<factor>]} = NU$
4.1	<term>	::=	<factor>	NU	$addr_{[<term>]} = NU$
6.1	<variable>	::=	$\mathcal{V}$	Y	$addr_{[<variable>]} = Y$
5.1	<factor>	::=	<variable>	Y	$addr_{[<factor>]} = Y$
4.1	<term>	::=	<factor>	Y	$addr_{[<term>]} = Y$
3.1	<arith expr>	::=	<term>	Y	$addr_{[<arith expr>]} = Y$
7.1	<integer>	::=	$\mathcal{J}$	15	$addr_{[<integer>]} = 15$
5.2	<factor>	::=	<integer>	15	$addr_{[<factor>]} = 15$
4.1	<term>	::=	<factor>	15	$addr_{[<term>]} = 15$
3.1	<arith expr>	::=	<arith expr> +<term>	Y + 15	CLA Y ADD=15 STO t_1
5.3	<factor>	::=	(<arith expr>)	(Y + 15)	$addr_{[<factor>]} = t_1$
4.1	<term>	::=	<term>*<factor>	NU*(Y + 15)	LDQ NU MPY t_1 STQ t_2
3.1	<arith expr>	::=	<term>	NU*(Y + 15)	$addr_{[<arith expr>]} = t_2$
2.1	<assignment>	::=	<variable>= <arith expr>	X = NU*(Y + 15)	CLA t_2 STO X

Given this mechanism, which we shall refer to in the sequel as the "Direct Generation Mechanism," plus some mechanism for creating and housekeeping local or internal labels and a sufficiently sophisticated assembler (e.g., it can allocate memory for constants and variables) we have a rudimentary compiler.

TABLE 5

Machine Code Semantics for Syntax II—Deferred Generation

1.1	<program>	::= <assignment> [1]	1. Output END
1.2		| <assignment> ; <program> [1] [2] [3]	
2.1	<assignment>	::= | <variable> = <arith expr> [1] [2] [3]	1. Process [1] 2. Process [3] 3. Output CLA addr [3] 4. Output STO addr [1]
3.1	<arith expr>	::= <term> [1]	1. Process [1] 2. Set addr = addr [ ] [1]
3.2		| <arith expr> + <term> [1] [2] [3]	1. Process [1] 2. Process [3] 3. Output CLA addr [1] 4. Output ADD addr [3] 5. Output STO addr [ ]
4.1	<term>	::= <factor> [1]	1. Process [1] 2. Set addr = addr [ ] [1]
4.2		| <term> * <factor> [1] [2] [3]	1. Process [1] 2. Process [3] 3. Output LDQ addr [1] 4. Output MPY addr [3] 5. Output STQ addr [ ]
5.1	<factor>	::= <variable> [1]	1. Process [1] 2. Set addr = addr [ ] [1]
5.2		| <integer> [1]	1. Process [1] 2. Set addr = addr [ ] [1]
5.3		| (<arith expr>) [1] [2] [3]	1. Process [2] 2. Set addr = addr [ ] [2]
6.1	<variable>	::= $\mathcal{V}$	1. Set addr = the variable name recognized at this point of the input string. [ ]
7.1	<integer>	::= $\mathcal{I}$	1. Set addr = a symbolic name for the address in which will be kept the integer constant recognized at this point in the input string. [ ]

There is an interesting variation in this method of generating machine code. Let us suppose that, for one reason or another, it is desirable to complete the analysis for an entire <assignment> and produce the tree representation of its syntax, and then to generate the machine coding which is to correspond to the tree so constructed. Table 5 illustrates this approach. It is essentially Table 4 to which some further actions have been appended. In this table the action denoted by a bracketed numeral preceded by the word "process" is interpreted: "do the actions for the component indicated." Thus, again given the assignment:

$$X = NU^*(Y + 15)$$

an analysis of this assignment with respect to Syntax II could be carried out resulting in the tree of Figure 2. Given this tree, the actions indicated in Table 5 could result in the following:

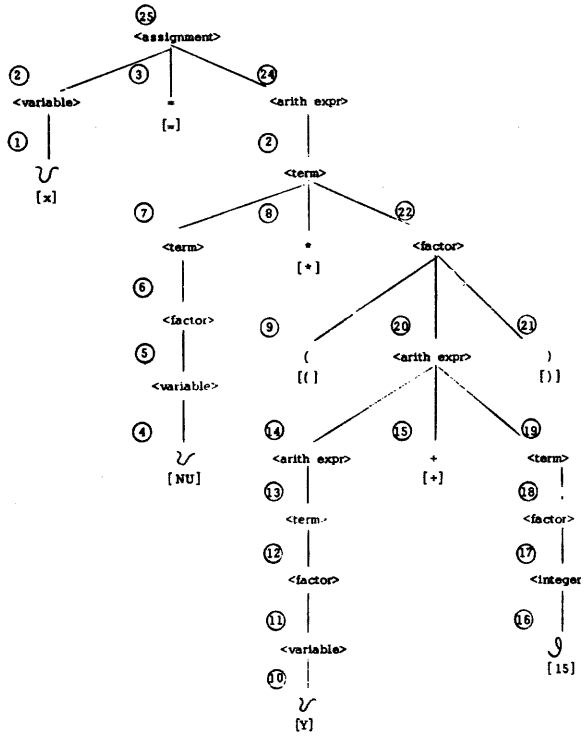


Figure 2. Syntax Tree for the <Assignment> "X = NU* (Y + 15)."

Tree Node	*Line of Table 5*	*Result*
25	2.1	Process <variable>, node 2.
2	6.1	Set $addr_{(2)} = X$; actions complete for node 2, return to node 25.
25	2.1	Process <arith expr>, node 24.
24	3.1	Process <term>, node 23.
23	4.2	Process <term>, node 7.
7	4.1	Process <factor>, node 6.
6	5.1	Process <variable>, node 5.
5	6.1	Set $addr_{(5)} = NU$; actions for node 5 complete; return to node 6.
6	4.1	Set $addr_{(6)} = NU$; actions for node 6 complete; return to node 7.
7	4.1	Set $addr_{(7)} = NU$; actions for node 7 complete; return to node 23.
23	4.1	Process <factor>, node 22.
22	5.3	Process <arith expr>, node 20.
20	3.2	Process <arith expr>, node 14.
14	3.1	Process <term>, node 13.

Tree Node	*Line of Table 5*	*Result*
13	4.1	Process <factor>, node 12.
12	5.1	Process <variable>, node 11.
11	6.1	Set $addr_{(11)} = Y$; actions for node 11 complete; return to node 12.
12	5.1	Set $addr_{(12)} = Y$; actions for node 12 complete; return to node 13.
13	4.1	Set $addr_{(13)} = Y$; actions for node 13 complete; return to node 14.
14	3.1	Set $addr_{(14)} = Y$; actions for node 14 complete; return to node 20.
20	3.2	Process <term>, node 19.
19	4.1	Process <factor>, node 18.
18	5.2	Process <integer>, node 17.
17	7.1	Set $addr_{(17)} = =15$; actions complete for node 17, return to node 18.
18	5.2	Set $addr_{(18)} = =15$; actions complete for node 18, return to node 19.
19	4.1	Set $addr_{(19)} = =15$; actions for node 19 complete; return to node 20.
20	3.2	Output CLA Y Output ADD =15 Output STO t_1 Set $addr_{(20)} = t_1$; actions for node 20 complete; return to node 22.
22	5.3	Set $addr_{(22)} = t_1$; actions for node 22 complete; return to node 23.
23	4.2	Output LDQ NU Output MPY t_1 Output STQ t_2 Set $addr_{(23)} = t_2$; actions for node 23 complete; return to node 24.
24	3.1	Set $addr_{(24)} = t_2$; actions for node 24 complete; return to node 25.
25	2.1	Output CLA t_2 Output STO X Set $addr_{(25)} = X$; actions for node 25 complete; exit.

Note that we have changed notation slightly and used the notation $addr_{(5)}$, for example, to indicate the (result) address which is associated with node 5.

The "code generation" mechanism implied by the above description is as follows: At any point in time some action of some node is to be performed; the actions and their interpretations are:

Action	*Example*	*Interpretation*
Process a node	Process [1]	Remember which action of the current node is being processed; perform the first action of the indicated node.
Output code	Output CLA addr [1]	Procure the result address from the node indicated (the node corresponding to addr [1] in the example) and output the instruction. If the address indication is blank (e.g., STO addr [ ]) select the next available temporary register address; set that address as the result address of the current node.
Set address	Set addr [ ] = addr [2]	Set the result address of the current node as the result address of the node indicated.
End of actions		Return to the next action of the node which "called" for this node to be processed.

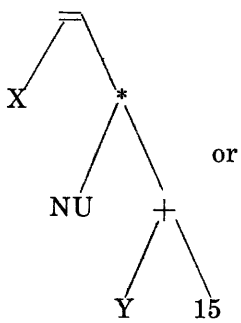
Thus, the mechanism has a "control" element which is, at any point in time, considering some node. When a new node is to be processed, the "control" remembers which action to proceed with when "control" returns to the current node and then initiates the first action of the node to be processed. When all actions for a node are processed, "control" returns to the node which called for the current node and takes up actions for that node where it left off. Further, the mechanism has the ability to output code, associate result addresses with the node being processed, and procure temporary register addresses.

Again given this mechanism, which we shall refer to in the sequel as the "Deferred Generation Mechanism," plus some mechanism for creating and housekeeping local or internal labels and a sufficiently sophisticated assembler, we have a rudimentary compiler.

There are, of course, other kinds of "actions" one could associate with a node. For example, it would be quite straightforward to associate actions for producing a different tree structure than the complete syntax tree as depicted in Figure 2. This might then produce, from an analysis of the assignment

$$X = NU*(Y + 15)$$

the simple tree (or "Polish prefix" representation):



or (=(X, *(NU, +(Y, 15))))

It will be apparent that the Direct Generation Mechanism does not require that the complete syntactic tree actually be built as the analysis proceeds. Rather, it is sufficient that there be some means (for example, a push down store is adequate) for "remembering" the result addresses which have yet to be "used." Further, while this technique appears, on the face of it, to be quite rapid and efficient (no tree need be kept, shorter "driving tables"—compare Table 4 with Table 5) it is subject to some serious limitations. In particular, since the coding for a syntactic type is actually being output as that type is recognized, there must (for most languages) be some mechanism for "erasing" a patch of code generated for a syntactic type recognized while the analyzer was attempting recognition of some larger construction when it turns out that the syntactic type in question does not enter into the construction of the larger syntactic type as it is finally built.

In the above example with the Deferred Code Mechanism we used <assignment> as the syntactic type over which (i.e., over the tree representation of which) generation was to occur. It is, of course, possible to generalize this to allow any syntactic type to be "tagged" as big enough to call the generation mechanism. Thus, at one extreme a complete program would have to be recognized before any generation was performed (a "two pass" compiler) and, at the other extreme, each syntactic type would call for generation ("one pass" compiler) thus making the Deferred Generation Mechanism essentially the same as the Direct Generation Mechanism. It should be noted that, employing the Deferred Generation Mechanism, once the tree corresponding to some syntactic type has been processed ("generated over") it can be erased with the exception of its top-most or root node which may have to remain to supply a "result address" for the generation over some larger construction.

It must be emphasized that both these mechanisms are very rudimentary and for use within a compiler would require some embellishment in order to be practical. Thus, for example, it seems rather a shame to generate a complete syntax tree for some (perhaps fragment of some) program and then make essentially no use of the contextual information contained implicitly in the tree structure. Indeed, a rather simple addition to make some use of this information would be the following: consider that we add conditional actions of the following sort:

IF addr [a] = addr [b], SKIP n
IF addr [a] ≠ addr [b], SKIP n
SKIP n

where in the first two the truth of the relation causes the n actions following to be skipped and the SKIP n action causes the n actions following to be skipped. If we further add "AC" and "MQ" as special values for addr [ ], then for a single address computer (say, like the IBM-7094), it would be possible to generate rather more efficient coding by placing results temporarily in the accumulator (AC) or multiplier quotient (MQ) registers and then checking for the use of these locations for operands before generating coding. Thus we might then associate with the construction <arith expr> ::= <arith expr> + <term> in Table 5 the actions:

1. Process [1]
2. Process [3]
3. IF addr [3] = AC, SKIP 4
4. IF addr [1] = AC, SKIP 1
5. Output CLA addr [1]
6. Output ADD addr [3]
7. SKIP 1
8. Output ADD addr [1]
9. Set addr [ ] = AC

These would have the effect of preserving results of additions in the accumulator and remembering that they were there in order to avoid redundant store-load instructions. In order to fully utilize such a facility, including keeping track of the MQ contents as well, some further mechanism for indicating the AC or MQ are empty or full and some mechanism for storing their contents would be required. The basic scheme is, however, reasonably clear from the example. The MAD Translator has facilities similar to these built into its code generation mechanism.

Even such a mechanism as this barely makes use of the rather rich store of contextual information available. In order to do so, however, we would require some means for talking about the nodes of the tree relative to any given node of interest (such as a nodes "father," "father's father," "father's right-brother's son's son," and so on). Further, it would probably be desirable to extend the "control" some what and allow more general "tree walks" than simply processing "sons" and returning to "father." Also, if contextual information were gathered, it would have to be "parked" somewhere and thus an addition of further information fields associated with each node would be useful plus, perhaps, some "working variables" which various actions could reference and set.

It is clear that one could extend the whole generation mechanism to include control sequencing and actions which were programs in a rather specialized programming language. The paper by Warshall and Shapiro [1] contains a brief sketch of one such extension which has been successfully tried on several computers and for several languages.

SOME PROBLEMS AND LIMITATIONS

The techniques and fragments of techniques which we have discussed above are all, in one way or another, relevant to the design and construction of compilers. Furthermore, these techniques, in the simplified form in which we have presented them, are of sufficient general utility that they should be in every "systems" programmer's bag of tricks. For example, the ANALYZE algorithm—coupled with either the Direct or Deferred Generation Mechanism discussed in the preceding sections—can be applied to a variety of programming tasks imbedding simple algebraic facilities in an assembly program, handling the "translation" of free format or formatted control cards, interpreting descriptions of formatted data, and so on. However, for the serious construction of compilers these techniques represent only a few of the techniques required and they are subject to some limitations.

Some of the considerations which must enter into any compiler design and which are affected to one degree or another by the choice of method of analysis and its linkage to the generation of code are the following:

—error analysis and recovery
—analysis of languages in which recognition order is not the same as generation order
—processing declarations
—generation of highly efficient coding

Let us consider these questions.

Error Analysis and Recovery

An error is detected, for example, in Syntax II, when the analyzer cannot recognize a <program>. Although the exact point in the input string past which recognition fails will be known, it is extremely difficult to determine exactly *why* the error occurred and to, in a general way, devise means for recovery.

Several schemes exist for dealing with this problem, notably:

1) A scheme which permits specification of "no back up" on certain constructs. For example, in Syntax II, no back up on recognition of "=" or "(" could help isolate the reasons for a failure.

2) A scheme due to E. T. Irons [5] which, in effect, carries along all possible parses of an input string.

3) Special "error" syntactic types which could be defined in the syntax.

At the present time there is no completely satisfactory scheme for dealing with syntactic errors discovered in the course of predictive analysis. If the programming language which is being analyzed has sufficiently simple structure that it is a *precedence grammar*, the technique of bounded context analysis is probably a better technique to utilize. A discussion of precedence grammars is given in Reference [6]; the use of bounded context analysis is described in Reference [7].

Recognition Order Differs from Generation Order

Some reasons why the order of generation might be different from the order of recognition are:

1) The detection of, and generation of coding for, sub-expressions which are common to two or more parts of the program is desired.

2) The detection of computations which are invariant in some larger computation (for example within loops) is desired.

3) Languages other than the usual programming languages are being translated, for example, data description languages or the computational fragments associated with TABSOL-like descriptions are to be processed.

Reference [1] describes some techniques for coping with these problems in a compiler which uses predictive analysis.

Handling Declarations

Here the problem is that the "actions" are not to generate coding (usually) but to change the syntax—normally through type coding information inserted into a symbol table. Formally, however, a declaration of type is really the appending of a syntax rule. Thus the ALGOL 60 declaration

"*real* X, Y;"

means that the two new syntax rules

"<real var>::=X"
and
"<real var>::=Y"

must be appended to the syntax.

Other declarations may cause changes to the full compiler—for example, debug mode declaration, and the like.

Generation of Highly Efficient Coding

This cannot be accomplished by generating code directly as the analysis is performed since common sub-expressions, invariant computations and the like couldn't be detected reasonably and special registers such as index registers certainly couldn't be allocated on any global basis, which is necessary if any really effective use is to be made of them.

Many of the manipulations required to collect the information pertinent to optimizing code are not particularly easily done (or, at least efficiently done) with the source material in a syntax tree form. Reference [1] describes a method by which such optimizations are performed over a set of "macro-instructions" which are "generated" by a technique similar to that depicted by Table 5.

SUMMARY AND CONCLUSION

In this paper we have tried to explain the workings of syntax-directed compiling techniques—or perhaps better, of those parts of a compiler in which the actions to be performed can reasonably be associated with the structure of the input string. A satisfying understanding of the operation of a syntax-directed analyzer can only be attained by actually playing through a few examples. We recommend this as a worthwhile experience to anyone who is interested, and so we have given a sufficiently detailed description of a particular example to permit the reader to write statements of his own in the simple language, and play them through the Analyzer and any one of several code-generation techniques.

There remains the question of evaluating syntax directed compiler techniques in comparison to other approaches.

On the face of it, syntax directed analyzers cannot be as efficient as operator-precedence techniques for the simple task of recognizing input structures. This follows from the fact that, no matter how cleverly the language designer, or specifier, arranges the elements of his Syntax Specification, the Analyzer will necessarily spend some percentage of its time exploring blind alleys. Clever specifications can make the blind alleys less frequent and shorter, but even for the simplest of languages, there will be some.

Thus, in any situation where the primary consideration is speed of the compiler itself, syntax-directed techniques are not the most suitable. But this, we argue, is true only if the quality of the coding produced is also of relatively little importance. In our experience with attempts to generate highly efficient optimized coding for several different machines, we find that the time spent in analyzing is a small fraction of the total—even using very sloppy Syntax Specifications. The most important question in compiling system design today, we reiterate, is *not* the "understanding" of the source language—that is a solved problem—but rather the generation of really good object-language coding.

One of the principal arguments in favor of syntax-directed techniques is that it is very easy to change the specification of the language, or, indeed, to switch languages, merely by changing the Syntax Tables—no modifications of the algorithms are required. And this is in fact true, with some restrictions. But now that techniques exist for automatically producing operator-precedence tables from a Syntax Specification [6], the syntax-directed compilers no longer have a monopoly on this useful feature.

A further advantage of syntax-directed analysis remains, up to the present, only potential. These techniques are evidently not of the "bounded context" sort—a syntax directed Analyzer can take into account as large a context as required to perform its recognition (admittedly, at a definite cost in speed). Hence, when the day comes that we need to perform analysis of source languages of much less rigid structure, syntax-directed techniques will be more immediately applicable than the techniques which are designed to take advantage of the restrictive properties of present programming languages.

In summary, syntax-directed techniques have a definite place in today's computing world, and promise to play an even more important role in the future.

APPENDIX

Some Further Transformations of the Syntax Specification and the Syntax Tables

In constructing the Syntax Tables, we described a complicated operation for avoiding the problem of left-recursive Alternatives in a Syntactic Type Definition. We can describe this as a transformation within the Syntax Specification itself, and, at the same time, include some features which improve the efficiency of the encoding of the Syntax.

First, we extend the idea of "Component" to include two new forms:

1) An Alternation (of one or more Constructions), enclosed in square brackets '[' and ']'. These brackets are assumed to be different from any of the Terminal Characters (if they were not, we'd use some other signs).

2) An Alternation enclosed in braces '{' and '}', again assumed different from any of the Terminal Characters.

Second, we apply a left-distributive law:

(T2) Whenever, within a single definition, two (or more) Alternatives start with the same Component (or sequence of Components), replace all of these Alternatives with a single one, whose last Component is the bracketed Alternation of the non-common parts of the original Alternatives, and whose first Component(s) is (are) the one(s) common to the original Alternatives. It is also useful to introduce the idea of a "null" Syntactic Type—effectively a Terminal Type which is always recognized whenever it is called for—denoted here by a capital lambda. Then, for example:

<a>::=<b> <c> <d> <e> |
<b> <c> | <b> <c> <f> | <g>

would be transformed into:

<a>::=<b> <c> [<d> <e> |
<f> | Λ] | <g>

(Obviously, if the Analyzer is going to consider Alternatives in the order in which they are written, a null Alternative should always appear last in an Alternation.)

Having applied (T2) to any Definition, there can be at most one left-recursive Alternative, and if there is one, we can rewrite the definition according to:

(T3) Put the left-recursive Alternative as the last Alternative in the definition; if there is more than one other Alternative, put square brackets around the Alternation consisting of the nonrecursive Alternatives; delete the sign ' ' preceding the last Alternative, and delete the first Component of that Alternative (which will be the same as the Defined Type of that Definition); then enclose the remaining Components (of this formerly last Alternative) in braces.

Thus the ultimate transform of a left-recursive definition has as Definiens a single Construction, the last Component of which is "iterated" (enclosed in braces). As an example, a Definition which was originally:

<a>::=<b> <c> | <a> <d> <e> |
<a> <f> | <g> <h>

would be transformed into:

<a>::= [<b> <c> | <g> <h>]
{<d> <e> | <f>}

The modifications to the rules for constructing the Syntax Tables to represent Definitions in this form is left as an exercise for the reader.

The Analyzer flow-charted in Figure 1 should work on the resulting tables.

Three of the Definitions in Syntax II would be changed by application of (T2) and (T3):

<program>::=<assignment>
[; <program> | Λ]
<arith expr>::= <term>
{+<term>}
<term>::= <factor> {* <factor>}

Now, an analogous pair of transformations—a right-distributive law, and the elimination of right-recursive Constructions—could be applied, and this would render the Syntax Specification still more compact. The language used by Brooker and Morris [10] for specifying syntax is essentially one of this sort, although the notation used is rather different.

More "Groundedness" Problems

The treatment we have described takes care of left-recursive definitions, as long as the recursion occurs within the definition of a single Syntactic Type. It will not handle the infinite-loop problem engendered by, as an example:

<a>::=<b> Z | X
<b>::=<a> Z | Y

and it is in general true that, for an Analysis technique of the "top down" sort, as presented here, a Syntax Specification with such left-recursive loops will not be adequate. This leads to the requirement of an additional condition on Syntax Specifications: If we say that a Construction is "firmly grounded" when its first Component is either a Terminal Type or a firmly grounded Defined Type, and a Defined Type is firmly grounded when *all* of its non-left-recursive Definientes (in the Basic Syntax Specification) are firmly grounded, then:

(C4) Every Defined Type must be firmly grounded.

In practice, this is not a serious restriction. The simplest test for this condition is to try to run the Analyzer—it stops requesting input and goes into a loop. It is usually a simple matter to rewrite the Syntax Specification to eliminate the difficulty. In the above example, this could be done in several ways, one of which is:

<a>::= <b>
<b>::= [X | Y] {Z }

A Modified Analyzer

The Analyzer algorithm of Figure 1 is designed to call the Generator upon recognition of *every* instance of a Syntactic Type, even if it is not the "longest" instance of that type present at the given position of the Input string. It turns out to be the case that, for all the standard programming languages, when the Analyzer needs to recognize a recursively defined Syntactic Type, it wants the longest string which is a member of that Type—that is, it should keep re-entering the iterated Component of the Definition (in our latest transformed form) until it meets a failure. The Syntax Tables and Analyzer described in this paper will find the "longest" instance of a type but this Analyzer does report each partial recognition also.

Now, a slight change in the Analyzer algorithm allows it to avoid reporting partial recognitions to the Generator, and call it only when it has completed recognition of the longest instance of a Syntactic Type. For those who might be interested in exploring this point, the changes to be made are:

1) Eliminate the boxes (GENERATE) and (STRUCT[SOURCE])? from the flow chart.

2) Insert a (GENERATE) box between the boxes [Popup GOAL, SOURCE] and (SOURCE=0?)

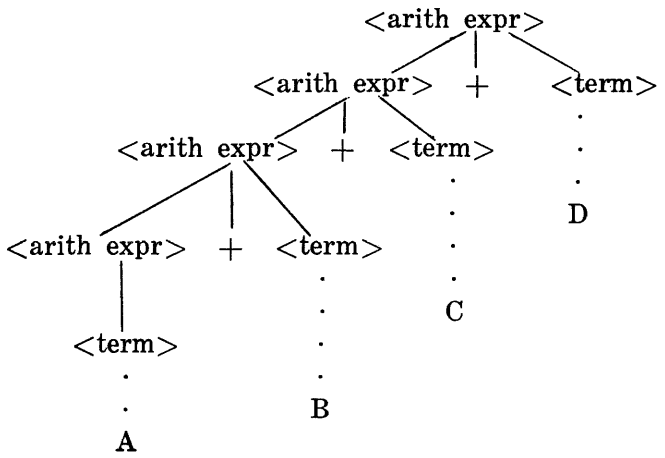
The entire Syntax Structure Table entry STRUCT can also be eliminated.

In order to correctly record the recognitions, the Generator must construct a slightly different tree (we are here assuming operation in the "deferred generation" mode), the form of which is best illustrated by an example:

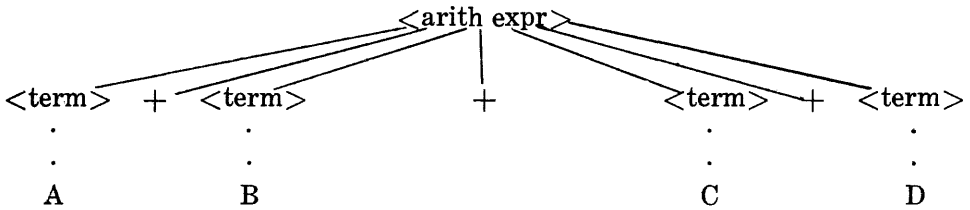
For the Input (sub-) string,

A + B + C + D

the Generator discussed in this appear will produce a (sub-) tree:



If the modifications mentioned above are made in the Analyzer, each of the terms in the input string will cause the Analyzer to signal recognition of a <term>, but only one recognition of <arith expr> will be reported, and that after all four <terms>. The Generator will then tree this recognition as follows:



To use a tree of this form for the generation of output code, the Generator language must be extended, for example, to refer to "siblings" instead of just "sons" and "fathers" and also to "count" sons (the number per node is not fixed, but depends upon the actual input) or to recognize the non-existence of a "sibling," etc. Typically, the processing sequence for the above tree is for a <term> node first to send control to its first son, to evaluate the sub-tree, then when control returns, to send control "horizontally"—to its siblings—for evaluation of the other <term>s; only the last <term> node would send control back to the father. References [1] and especially, [2], discuss this in more detail.

A More Subtle Problem

The Analyzer algorithm given in this paper has the following property: Assume that a given Syntactic Type <x> has been recognized at character-position *c* in the input string, and the Analyzer fails to recognize any of the possible successors of <x> within the Definition of its current goal. The Analyzer will report failure for this Alternative of the Definition. It is possible to formulate a Syntax Specification in such a way that this reported failure would be erroneous: if there were another, different, substring starting at *c* and which was also an <x> (this other substring would correspond to an Alternative occurring *later* in the Definition of <x> than the Alternative which was first successfully recognized). It is certainly possible to design an Analyzer which will keep track of the information necessary to allow this kind of "back-up" (see References [5] and [13]), but for the present purposes it would have encumbered the description with a good deal of additional mechanism — essentially, the Syntax Structure Table would have another item, encoding the converse of the relation represented by SUCCESSOR, and push-down storage would be required to keep track of the SOURCE lines of the Types successfully recognized, instead of just those which are currently being worked on. Using the Analyzer and Generator described in this appendix, it becomes much easier to ac-

commodate this feature, since the required additional information can easily be kept in the tree while it is being built. Reference [2] discusses this question in detail and with examples.

"Top Down" vs. "Bottom Up"

The Analyzer described in this paper is of the sort known as "top down," the appellation referring to the order in which the Analyzer sets its goals. The present Analyzer will always set as its next goal some Syntactic Type which appears as a Component in the Definition of its present goal — any time the Recognizer is called to find a Terminal Type, the pushdown storage of the Analyzer will contain a record for each Syntactic Type in a chain reaching down from the Starting Type to the Terminal Type. The order in which a "bottom up" Analyzer sets its goals is much more difficult to describe, but the actions of the two types can be impressionistically sketched as follows:

The "top down" Analyzer sets a goal and tries all possible ways of achieving that goal before giving up and replacing the goal with an alternative.

The "bottom up" Analyzer, having recognized a Syntactic Type, checks whether it has "gone astray" in trying to reach its goal or whether that Type is indeed a possible first Component of a first Component of . . . of the goal. If the latter, it continues processing input until it has built another Type of which the previous one is a first Component, and goes back to the checking. If it has gone astray, it backs "down" and tries to see if it can construe the input differently, to approach its goal along a different chain of intermediate types.

E. T. Irons' original syntax-directed Analyzer design was of this type (Reference [4]). It might be interesting to characterize an Analyzer similar to Irons' within the terminology of this paper.

We start with the Basic Syntax Specification, and first build a magical matrix which will answer the question "Can α start with β?" where α and β are Syntactic Types. The relation "can start with" is defined recursively as follows:

A (Defined) Syntactic Type α can start with the Syntactic Type β either

1) if β appears as the first Component of some Definiens of α, or

2) if there exists a γ which can start with β, and γ occurs as the first Component of some Definiens of α. Irons' paper [4] gives an elegant technique for constructing this matrix. Note that α can start with α, if it is left-recursively defined, but not otherwise.

The next step is to transform the Basic Specification:

First, remove the Defined Type and the sign '::=' from the left-hand end of the Definition, and place '=::' followed by the Defined Type at the right-hand end. (In effect, when a definition is considered from left to right, the Type which it defines is not known until all the Components have been recognized.) Hereafter, the Defined Type will be called the "Result" of the Definition. For example, the Definition of <assignment> becomes:

<variable>=<arith expr>=::
<assignment>

Second, apply the left-distributive law to the set of definitions, introducing Alternation signs '|' as required. To illustrate, the following two interesting lines would result, in the example language of this paper:

<term> [*<factor> =::<term> |
=:: <arith expr>]
<variable> [= <arith expr>=::
<assignment> | =::<factor>]

The effect of this transformation is to reduce the set of definitions to one line for each Syntactic Type which occurs as a first Component of one of the original Simple Definitions.

From the resulting set of "definitions" syntax tables are constructed, analogous to the ones in this paper. But the analogue of the Snytax Type Table is now a directory of first Components, each entry of which points to the first of a block of structure-table entries which encode the remainder of the "definition," now including a mention of the Result of the original Simple Definition (suitably flagged to avoid interpreting it as just another successor).

For use with a "bottom up" Analyzer of this sort, the Terminal Types of the language (or at least those which appear as first Components in any Definition) must be unambiguously recognizable independently of context—that is, the Recognizer may be told merely to "find something," and it will return with an indication of the particular Terminal Type it recognized.

To start the analysis, the goal is set to the Starting Type, and the Analyzer proceeds as follows:

Step 1 Call the Recognizer; the Terminal Type it reports is placed in Type In Hand.

Step 2 Can the goal start with the Type In Hand? If not, go to Step 4. If so, proceed to Step 3.

Step 3 Consult the Structure Table at the point indicated in the Type Table for the Type In Hand. Push down the current goal and its source and set up as new goal the Component mentioned in this entry in the Structure Table. (This structure-table entry is the "source of this goal"). Go to Step 1.

Step 4 Is the Type In Hand the same as the goal? If not, go to Step 7. If so, proceed to Step 5.

Step 5 (We have attained a goal) Consider the source of this goal. Is the successor of that entry in the Structure Table flagged as a Result? If so, go to Step 6. If not, replace the goal with the (Syntactic Type mentioned in the) successor of the source, reset the source to point to this successor, and go to Step 3.

Step 6 (We have recognized all the Components of a Definition.) Place the name of the Type mentioned in the Result entry into Type In Hand, pop up the goal (and source), and go to Step 2.

Step 7 (We have "gone astray.") Consider the Structure Table entry for the source of the current goal. Does it have an alternate? If not, go to Step 8. If so, restore the input-string pointer to the value it had when the current goal was first set up, replace the current goal with the alternate (adjust source), and go to Step 1.

Step 8 Pop up the goal and source, and go to Step 4.

For programming languages of the current sort, there is no clear advantage in favor of either the top down or bottom up analysis techniques, insofar as efficiency of the Analyzer is concerned. For either technique, it is possible to design a language and Syntax Specification on which the technique will perform very poorly, while the other one will not be nearly as bad. The choice between the techniques is generally made on the basis of considerations other than raw speed of the analysis, such as the kind of output desired from the analysis, the possibility of error detection and correction, or personal taste.

"Bootstrapping"

As a final comment, we merely point out the fact that the language of the Syntax Specification is itself a rather straightforward, well-behaved language, easily susceptible of being described by a Syntax Specification. A version of the compiler can be written which uses a Specification of the Syntax-Specification-Language to "drive" it, and produces, instead of output code in a machine language, a set of Syntax Tables which encode the Syntax Specification it receives as input. This has, in fact, been done (References [1], [2]).

BIBLIOGRAPHY

1. WARSHALL, S., and SHAPIRO, R. M., *"A General Purpose Table Driven Compiler,"* to be published in the Proceedings, SJCC, Spring 1964.
2. SHAPIRO, R. M., and ZAND, L., *"A Description of the Input Language for the Compiler Generator System,"* CAD–63–1–SD, Computer Associates, Inc., June 1963.
3. CHEATHAM, T. E., JR., and LEONARD, GENE F., *"Introduction to the CL–II Programming System,"* CA–63–7–SD, Computer Associates, Inc., Nov. 1963.
4. IRONS, E. T., *"A Syntax Directed Compiler for ALGOL–60,"* Comm. ACM 4 (1961), 51–55.
5. IRONS, E. T., *"An Error Correcting Parse Algorithm,"* Comm. ACM 6 (1963, 669–674.

6. FLOYD, R. W., *"Syntactic Analysis and Operator Precedence,"* Jnl. ACM, vol. 10 (1963), p. 316.
7. GRAHAM, R. *"Bounded Context Translation,"* to be published in the Proceedings, SJCC, Spring 1964.
8. NAUR ET AL., *"Report on the Algorithmic Language ALGOL 60,"* Comm. ACM, vol. 3 (1960), p. 299.
9. BARNETT, M. P., *"Continued Operator Notation for Symbol Manipulation and Array Processing,"* Comm. ACM 6 (Aug. 1963), p. 467.
10. BROOKER, R. A., and MORRIS, D., *"An Assembly Program for a Phrase Structure Language,"* The Computer Journal, vol. 3 (1960), p. 168.
11. KUNO, S., and OETTINGER, A. G., *"Syntactic Structure and Ambiguity of English."* AFIPS Conference Proceedings, vol. 24 (1963).
12. IRONS, E. T., *"PSYCO, The Princeton Syntax Compiler,"* Institute for Defense Analysis, Princeton, N.J.
13. FLOYD, R. W., *"The Syntax of Programming Languages—A Survey,"* to be published in the IEEE Transactions on Electronic Computers.

A GENERAL-PURPOSE TABLE-DRIVEN COMPILER

Stephen Warshall and Robert M. Shapiro
Computer Associates, Inc.
Lakeside Office Park
Wakefield, Massachusetts

INTRODUCTION

If a compiler is to generate efficient object code, there are several different kinds of optimization which should take place. Each of these optimization procedures has a preferred domain: that is, some algorithms prefer to operate over the input string, others over the tree which describes the syntax of the string, others over the "macro-instructions" which are generated from the tree, and so forth. In an earlier paper,[1] one of the present authors pointed out the necessity for employing the tree form in particular as a natural domain for optimizers which consider syntactic context and suggested that, just as Irons[2] and others had built general-purpose table-driven parsing algorithms, one could also build a general-purpose table-driven program for getting from trees to macro-instructions. The final compiler design presented here is the result of pursuing that kind of thinking somewhat farther.

COMPILER ORGANIZATION

The compiler is composed of five phases (not "passes," if that means pulls of the input tape):

1. A syntactic *analyzer* which converts a piece of input string into a tree-representation of its syntax.

2. A *generator*, which transforms the tree into a sequence of n-address macro-instructions, investigating syntactic context to decide the emission.

3. An *"in-sequence optimizer"* (ISO) which accumulates macros, recognizes and eliminates the redundant computation of common subexpressions, moves invariant computations out of loops, and assigns quantities to special registers.

4. A *code selector* which transforms macros into syllables of machine code, keeping complete track of what is in special registers at each stage of the computation.

5. An *assembler* which simply glues together the code syllables in whatever form is required by the system with which the compiler is to live: symbolic, absolute, or relocatable, with or without symbol tables, etc.

The first four phases are encoded as general-purpose programs; the fifth has been handled as a special-purpose job in each version of the compiler, and will therefore not be covered in the present discussion.

Phase I: Analyzer

The analyzer is of the "top-down" syntax-directed variety, driven by tables which are in effect an encodement of the source language's syntax as given by a description in the metalinguistics of the ALGOL 60 report[3] (the so-called "Backus normal form"). There are several features of interest in this encodement: rules are shortened where possible by application of the distributive law (thus, "<A>

<B> | <A> <C>" would be coded as "<A> (<B> | <C>)"; it is possible to define types by naming an arbitrary scanner which recognizes them, thus eliminating syntax-chasing when forming ordinary identifiers, etc.; left and right recursive rules are carried in a transformed condition whose effect is to force recognition of the longest possible string which satisfies the rule. The analyzer tables contain some information which is not syntactic, but rather concerned with compiler control (how much tree should be built before the generator is called, for example) or with specifying additional information to be placed in the tree for later use by the generator.

Phase II: Generator

The generator algorithm "walks" through the tree from node to node, following directions carried in its tables. As it walks, macro-instructions are emitted from time to time, also as indicated in the tables. The encodement of a set of tables (a so-called "generation strategy") is based upon the idea of a "relative tree name." A relative tree name is, formally, a function whose domain is the set of nodes in the tree and whose range is its domain union zero. At any given time, the generator is looking at (has walked to) some particular node of the tree. Every relative tree name is then interpreted (evaluated) as a function of that node as independent variable. A relative tree name is a rule for getting from "here" to some neighboring node of the tree. Thus, if we may view the tree as a genealogical one, a relative tree name might "mean" father or first son or first son of second son, for example, of the node the generator is currently considering.

A generation strategy is composed of a set of rules each of which consists of a description of some kind of node which is somehow of interest together with a list of things to be done when a node of that kind is in fact encountered. A kind of node is generally distinguished by its own syntactic type and the types of some of its neighbors. The things to be done include walking to a neighboring node and emission of macros whose variables are neighboring nodes. In all cases, neighboring nodes are named in the tables by relative tree names.

Phase III: In-Sequence Optimizer

The ISO accepts macro-instructions emitted by the generator. The processing of a macro usually results in placing the macro in a table and sending a "result" message back to the generator. Macros also instigate various bookkeeping and control operations within the ISO.

The processing of a macro is controlled by a table of macro descriptions. A macro may or may not be capable of being combined with others into a common-subexpression; the arguments of a macro may or may not be commutable, and so forth. The ISO will detect macros whose arguments are literals and in effect execute those macros at compile time, creating new literals. If a macro may be handled as (part of) a common subexpression and is not computable at compile time, the ISO will recognige a previous occurrence of the same macro as equivalent if none of its arguments have been changed in value either explicitly or implicitly by any of the messages that have been received in the interval.

At some point the ISO receives a macro-instruction that signals an "end region" control operation. This causes the ISO to perform a set of "global" optimizations over the region of macros just completed. These global optimizations include the recognition of those computations which remain "invariant" within the region and the reservation of "special registers" such as index registers to reduce the number of special register loads and stores within the region.

Phase IV: Code Selector

The code selector produces symbolic machine code for a region of macros after the ISO has collected these macros and performed its various optimizations. The code selector is driven by a table of code selection strategy. The domain of a strategy is the region of macros and a "track table" which represents the condition of the registers of the target computer.

The code selector views the macros as nodes of a tree structure; that is, certain partial orderings exist which guarantee that the code emitted preserve the computational meaning of the macros, but within these constraints the

strategy can order the computation according to its convenience, as a function of the availability of registers and results. The preferred mode of operation is to postpone making decisions about what code to generate for a macro until it is known how the result of that macro will be used.

The code selector also makes use of certain information gleaned from the macro-description tables in order to predict how the special registers (index registers, etc.) will be used. These predictions enable the code selector to use such registers intelligently. This local optimization, combined with the global reservation of special registers by the ISO, results in a fairly effective use of these registers.

BOOTSTRAP TECHNIQUE

There are three major sets of tables to be prepared if the compiler is to be particularized to a specific source language and target machine. These are the syntax tables, the generation strategy tables, and the tables of macro description and code selection. The bootstrap technique is simply a method of automating part of the process of preparing these tables.

A group of three languages was developed, corresponding to the three tables. These languages are all describable in the Backus notation and thus capable of analysis by the analyzer. A set of syntax tables and generation strategy tables for these languages was encoded by hand and installed in the compiler, which was then capable of translating from these special languages into macro-instructions. The link from the generator to the ISO was broken and replaced by a link to a special "Bootstrap ISO" which converted these apparent macro-instructions into lines of tables, namely, the three tables of interest. Thus the process of table preparation was reduced to one of writing down statements in a family of readable languages.

THE LANGUAGE BNF

The syntax of the source language is described in a language called BNF (to suggest "Backus normal form," denoted B.n.f.). BNF looks much like B.n.f. to within the limitations of the available character set and a desire for legibility. Thus syntactic type names are given as identifiers (upper case alphanumerics), the sign "::=" is replaced by "=", and the sign "|" by "/"; literal symbol strings are headed by "$" and delimited by "/" or blank. Within a symbol string "$" acts as a control character and forces inclusion of the next character as part of the string. Thus, as long as the symbol strings do not include "$", "/", or blank, everything is quite readable; if they do, legibility drops accordingly.

Examples:

B.n.f.	BNF
<arex> ::= <term> | <arex> <adop> <term>	AREX = TERM/AREX ADOP TERM
<relop> ::= GE | LE | UE | EQ	RELOP = $GE/$LE/$UE/$EQ
<mulop> ::= *|/	MULOP = $*/$$/

To each BNF type definition may also be appended tags which provide control information associated with the type.

THE LANGUAGE GSL

A statement in the generation strategy language GSL begins with a predicate to be satisfied, of the form:

$$\text{IF } \langle\text{type name}\rangle \text{ AND } \mathcal{S}_1\ (t_1) \text{ AND } \ldots \text{AND } \mathcal{S}_n\ (t_n),$$

where the $\mathcal{S}_i$ are assertions whose truth is to be tested and the t_i are relative tree names. The $\mathcal{S}_i$ are assertions about the presence or absence of a node, its syntactic type, the number of its "sons" (components), and so on.

Following this predicate is a sequence of commands to be associated with nodes of the distinguished kind. A command is either a directive to proceed to another node or an action of some sort (emit output, for example).

Example:

IF AREX AND SON3 = 0 AND
FATHER*FATHER*RTSIB IS ADOP,
$LFTSIB $SON1 $OUTPUT
(PLUS,SON1,FATHER*RTSIB).

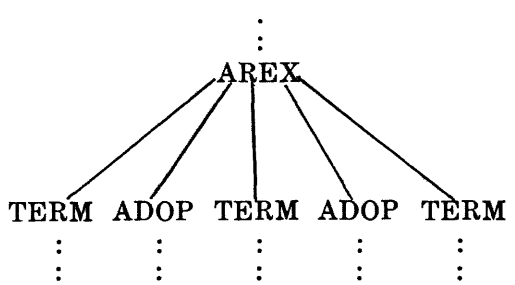
The list of commands is in effect a small program to be attached to a node when its kind is first determined. It is an odd program, in that each node keeps its own program counter. The generator may consider a node several times. The first time, its kind is determined, its generation strategy "program" attached, and the first command of the program executed. Upon subsequent reconsiderations of that particular node, the first command not yet executed is in fact performed. There are appropriate procedures for handling ill-defined commands (if a command directs walk to a non-existent node, take the next command; if there are no more commands, go to FATHER, and so on).

In the above example, upon first entry, the generator will just move consideration to the "left sibling" of the current node; at second entry, to the first son; at third, a PLUS macro will be emitted; afterward, any entry will cause automatic transfer to the father of this node.

Obviously, for a language of this sort—built around the relative tree name idea—to work at all, it is essential that the GSL programmer know in precise detail what tree structure will be output by the Analyzer. Thus the components of a type as given in the BNF and the sons of a node are in perfect one-one correspondence. The single exception is that left and right recursives come out as a "bush" of components at the same level; thus,

AREX = TERM/AREX ADOP TERM

would produce, from "A+B+C", a bush of the form



This exception is readily detectable by the BNF/GSL programmer, and thus does not violate the general philosophy.

The language GSL includes a number of commands besides "OUTPUT." These are concerned with passing information from node to node of the tree and with the emission of control messages to the ISO.

THE LANGUAGE MDL

The Macro Description Language provides a means of completely defining the semantics of the macro-instructions emitted by the generator. The definition contains the information necessary for controlling the processing of a macro by the ISO and the code selector, including a definition of the instruction repertoire and special registers of the "object" computer.

The most significant part of MDL is the sublanguage CSL (Code Selection Language). CSL permits the writer of a strategy to describe decision rules for the selection of machine instructions in terms of the status of macros (and their dependents: arguments, arguments of arguments, etc.) and the status of computer registers.

To illustrate a few of the properties of CSL and the code selector, assume the compiler is translating an algorithmic language with simple arithmetic assignment statements and has analyzed and generated over the statement

A ← B * C + A

producing the following macros:

(1) *, B, C
(2) +, (1), A
(3) ←, A, (2)

Assume that the computer has an accumulator (ACC), an addressable memory (M) and five single-address instructions:

CLA M : (M) → ACC
ADD M : (ACC) + (M) → ACC
MPY M : (ACC) * (M) → ACC
RAD M : (ACC) + (M) ACC, M
STO M : (ACC) → M

Following the principle that no code should be generated for a macro until the intended use of the result of the macro is known, the code selector would begin execution of the selection

strategy for macro line (3), the strategy for a "←" macro.

For this example the strategy would recognize that the RAD instruction was applicable and, after deciding on the execution of macro line one and verifying that its result is in the ACC, would emit an RAD A instruction.

The strategy for the "*" on macro line one would produce CLA B followed by MPY C.

Hence the code produced would be

```
CLA B
MPY C
RAD A
```

To give some idea of the appearance of a code selection strategy we append a set of statements in CSL. Each line has been numbered to facilitate referencing a set of explanatory notes. Also refer to the descriptions of EXECUTE, OUTPUT, and ALLOCATE which appear immediately after the strategies.

In the notes, the following symbols are used:

$\mathcal{V}$ variable
$\mathcal{E}$ expression
ω operation (either "+" or "x")

```
      FOR "←"
1.                IS ARG2 = "+" MACRO
2.                begin IS ARG1 (ARG2) = ARG1
3.                    begin  C1 = ARG2(ARG2)
4.                           GO TO ALPHA                    end
5.                    IS ARG2 (ARG2) = ARG1
6.                    begin C1 = ARG1 (ARG2)
7.                           ALPHA .. EXECUT*(C1)
8.                           ALLOCATE* (C1 TO ACC)
9.                           OUTPUT* ((RAD) ARG1)
10.                          EXIT                       end end
11.               EXECUT (ARG2)
12.               ALLOCATE (ARG2 TO ACC)
13.               OUTPUT ((STO) ARG1)
14.               EXIT
```

Notes for "←"

1. Is the second argument (i.e., the value being assigned to the first argument) the result of a "+" macro?
2. If so, is the first argument of the "+" macro identical to the variable receiving the assignment; in other words do we have the form $\mathcal{V} \leftarrow \mathcal{V} + \mathcal{E}$?
3. If so, the local variable C1 is set to the macro line number for $\mathcal{E}$.
4. And transfer control to ALPHA (line 7).
5. Alternatively, is the second argument of the "+" macro identical to the variable receiving the assignment; in other words do we have the case $\mathcal{V} \leftarrow \mathcal{E} +$?
6. If so, the local variable C1 is set to the macro line number for $\mathcal{E}$.
7. ALPHA.. Execute the code selection strategy appropriate for the macro on the line specified by C1 (i.e., cause the evaluation of $\mathcal{E}$.
8. Execute the code selection strategy subroutine ALLOCATE to guarantee that the result of macro line C1 (i.e., the value of $\mathcal{E}$) is in the accumulator.
9. Output to the assembler RAD $\mathcal{V}$.
10. Exit.
11. Otherwise (the fail path of 1. or 5.) execute the code selection strategy appropriate for the macro pointed to by the second argument. (We have the case $\mathcal{V} \leftarrow \mathcal{E}$ and wish to cause the generation of code to evaluate $\mathcal{E}$).
12. Execute the code selection strategy subroutine ALLOCATE to guarantee that the value of $\mathcal{E}$ is in the accumulator.
13. Output to the assembler STO $\mathcal{V}$.
14. Exit.

FOR "+"

1. C2 = (ADD)
2. BETA . . EXECUTE (ARG1)
3. EXECUTE (ARG2)
4. IS ARG2 IN ACC
5. begin C1 = ARG1
6. GO TO GAMMA end
7. ALLOCATE (ARG1 TO ACC)
8. C1 = ARG2
9. GAMMA . . OUTPUT ((C2) C1)
10. EXIT

Notes for "+"

1. The local variable C2 is set to the operation code for "ADD."
2. BETA . . Execute the code selection strategy appropriate for the macro pointed to by the first argument (we have the case $\mathcal{E}_1 \omega \mathcal{E}_2$, and wish to cause the generation of code to evaluate $\mathcal{E}_1$).
3. Execute the code selection strategy appropriate for the macro pointed to by the second argument (i.e., $\mathcal{E}_2$).
4. Is the value of $\mathcal{E}_2$ now in the accumulator?
5. If so, set local variable C1 to $\mathcal{E}_1$.
6. And transfer control to GAMMA (line 9).
7. Otherwise execute the code selection strategy subroutine ALLOCATE to guarantee that $\mathcal{E}_1$ is in the accumulator.
8. And set local variable C1 to $\mathcal{E}_2$.
9. GAMMA . . Output to the assembler the operation code specified by C2 and the address specified by C1.
10. Exit.

For "*"

1. C2 = (MPY)
2. GO TO BETA

Notes for ""*

1. The local variable C2 is set to the operation code for "MPY".
2. And transfer control to BTA (line 2 for "+" strategy).

EXECUTE (N): If N names a macro line this action causes the code selection to execute the strategy appropriate for the macro on line N and then resume where it left off.

OUTPUT (): This action outputs code to the assembler.

ALLOCATE (N_1 TO N_2): N_1 names a macro line and N_2 names a register (or register class). This code selection subroutine causes the code selection to guarantee that the value for which N_1 stands is placed in the register (or register class) named by N_2, saving the contents of the register if they are still needed.

A flow history of the processing for the example would be:

macro type	*line*	*resulting output*	
←	1		
←	2		
←	5		
←	6		
←	7		
*	1		
*	2		
+	2		
+	3		
+	4		
+	7	CLA	B
+	8		
+	9	MPY	C
+	10		
←	8		
←	9	RAD	A
←	10		

STATUS AND EVALUATION

This compiler and its associated bootstrap have been realized on a variety of machines (IBM 7090, Burroughs D-825, CDC 1604). The compiler has been used to translate from source languages JOVIAL, L_0 (the algebraic language of the CL-I System), and CXA (a BALGOL dialect) into several machine languages, including D-825 and 1604. The method of moving the compiler from machine to machine may be of interest as an indication of the power of the technique.

The compiler itself was originally written in language L_0 and compiled through the CL-I

Programming System into a running program on the IBM 7090. Then a deck of BNF/GSL/MDL cards defining the translation from L_0 into CDC 1604 was input to the bootstrap. After execution of the latter program, there existed a compiler for translating algorithms written in L_0 into an assembly language (meeting the requirements of the COOP system on the CDC 1604), operating on the IBM 7090. That compiler was fed the decks of cards in L_0 which had originally defined the compiler to CL-I. The result of this run was a compiler (and bootstrap) which could be moved to the CDC 1604. Then *that* version of the bootstrap was fed decks in BNF/GSL/MDL which defined the translation of CXA into CDC 1604, resulting in a CXA compiler on the CDC 1604.

The compiler is not as fast as some we have seen, but does not seem prohibitively slow, either. The object code is quite alarmingly good: indeed, it is frequently completely unreadable by a programmer of normal patience. In practice we prepare a BNF/GSL/MDL deck which does an adequate job. If we later want to improve the code (and are willing to slow down the compiler accordingly), we simply extend the deck.

The bootstrap method does not make compiler construction trivial, since code selection for a messy machine can be very difficult to work out and since contact with the data and control environment of the code being compiled may be more expensive than the translation process. What is now trival is the substantial modification of source syntax, minor changes in optimization rules, and the like.

The work described here has been informally reported to several agencies over the last year under various names, including "C.G.S." (for "compiler generator system"), the "bootstrap method," and the "COMPASS technique" for Computer Associates, Inc.).

BIBLIOGRAPHY

1. S. WARSHALL, "A Syntax Directed Generator", Proceedings of the EJCC, 1961, Macmillan and Co., 1961.
2. E. T. IRONS, "A Syntax Directed Compiler for ALGOL-60", *Communications of the ACM*, January, 1961.
3. NAUER (ed.) et al., "Report on the Algorithmic Language ALGOL-60," *Communications of the ACM*, Vol. 3, No. 5, May, 1960.

A COMPUTER TECHNIQUE FOR PRODUCING ANIMATED MOVIES

Kenneth C. Knowlton
Bell Telephone Laboratories, Incorporated
Murray Hill, New Jersey

INTRODUCTION

This paper describes a computer technique used for the production of animated diagram movies.* This technique—as implemented with the IBM 7090 computer and the Stromberg-Carlson 4020 microfilm recorder[1]—involves the basic steps of coding and checkout, production computer run, and optical printing from the master film thus produced.

Programs are coded in the "movie language" to be described, a language which has been developed entirely within the framework of MACRØ FAP.[2] They are checked out, without producing film, through examination of picture samples printed on the standard output printer.

After checkout, the production run produces a magnetic tape which instructs the 4020 in exposing a master film, such as that shown in Fig. 1. Each frame of this film is made of a rectangular array of tiny characters produced by the 4020 charactron tube used in the typewriter mode. The recording camera is slightly defocussed, thereby turning the finely structured characters into contiguous blobs of different intensities, depending upon the characters used. Each picture thus consists of a rectangular array of blobs on a raster either 126 wide and 92 high or, for finer resolution, 252 wide by 184 high. Figures 2a, 2b, and 3 of this paper were actually made by this system—operating in the fine-resolution mode—as if they were to appear as scenes in a movie.

The master film contains only one picture for each sequence of identical frames of the final movie; a comment above this picture indicates the length of the sequence. The "stretching out" of the master is done by optical printing at a movie laboratory, where standard processes are also used for editing, adding a sound track, and making work prints and final prints.

Internal Representation of Pictures

The movie programmer imagines that pictures exist within the 7090 on rectangular surfaces ruled off in squares, each square containing a number from 0 to 7. Pictures are created and manipulated by changing the patterns of numbers in the squares. During output these patterns of numbers are interpreted as spots of appropriate shades of grey, according to a programmer-specified transliteration. Fine-resolution pictures are produced by "aiming" the output routine at a subarea 252 squares wide and 184 squares high; for coarse-resolution pictures the output routine is aimed at an area 126 squares wide and 92 squares high.

*This computer technique is also described in a 17-minute 16mm black and white silent movie which was produced by the very process which it describes. This movie, entitled "A Computer Technique for the Production of Animated Movies", is available on loan from the Technical Information Libraries, Bell Telephone Laboratories, Incorporated, Murray Hill, New Jersey.

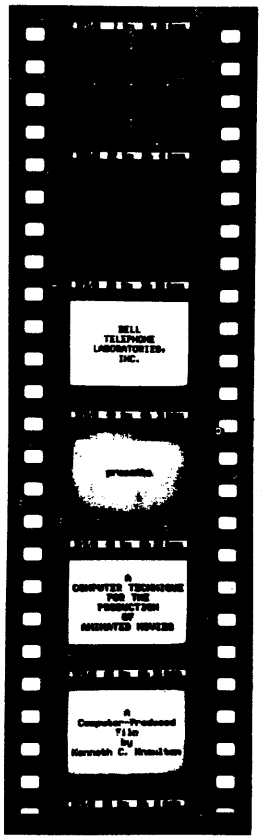


Figure 1. The beginning of a master film, shown here larger than actual size. The comment above each frame gives the date, the beginning and ending frame numbers of the corresponding sequence in the final film, and (redundantly) the length of this sequence. The final movie in this case begins with the framing and focusing pattern, followed by two seconds (48 frames) of black. Then the four title scenes appear, in order, followed by one second of black.

The total storage area within the 7090 corresponds to two complete fine-resolution movie frames. This area may be used in different ways: as two independent surfaces, each just large enough for one complete movie frame, or as one surface twice as wide or one surface twice as high. These possibilities are indicated in Fig. 2a which gives the names of these surfaces and their sizes in squares. Every square of a surface is assigned x and y coordinates, the bottom left-hand corner square of every surface having the coordinates x = 0, y = 0.

There are still other sizes and shapes of surfaces which can be used for complete coarse-resolution frames (or for parts from which fine-resolution pictures will be composed by copying). These surfaces, with their names and

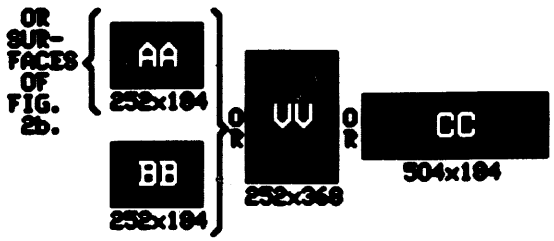


Figure 2a. Names and sizes of surfaces large enough to hold complete fine-resolution movie frames (252 x 184). Alternate uses of total 7090 storage area are indicated, including the uses of region AA in ways shown in Fig. 2b.

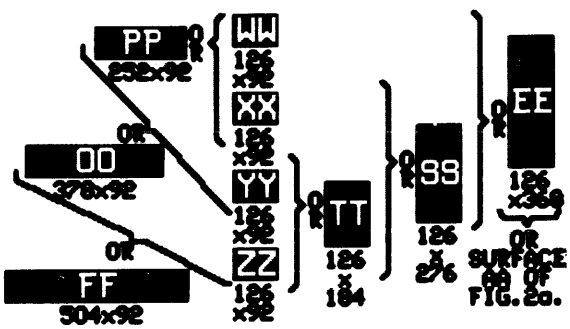


Figure 2b. Names and sizes of additional surfaces, each of which is at least large enough to hold a complete coarse-resolution movie frame.

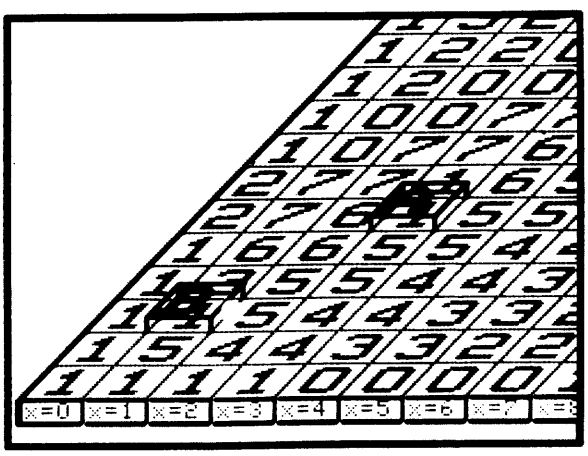


Figure 3. Representation of a surface with scanner A at x = 3, y = 5, and scanner B at x = 1, and y = 2.

sizes, are shown in Fig. 2b. Their use precludes the simultaneous use of surfaces AA, VV, or CC.

The Programming Language

The programmer's basic conceptual framework includes a number of scanners which he imagines to be sitting on various squares of the surfaces (see Fig. 3). Scanners can intercommunicate, and each can read the number it is sitting on, can write a new number into this square, and can move right, left, up, or down any integral number of positions to a new square. Scanners may also convey information —by virtue of their positions or the numbers on which they sit—to the subroutines of the movie system. There are 26 scanners in all, each named by a different letter of the alphabet.

The language by which the computer is programmed to make movies may be divided into two parts. The first and historically older part consists of those instructions for drawing and changing pictures by explicit manipulation of scanners. This part, the "scanner language," may be used for other purposes than movie making. It may be used, for example, to draw flowcharts or schematic wiring diagrams which are represented within the computer by two-dimensional arrays of the digits 0 through 7.

An example of an instruction in the scanner language is

IFANY (B,R,10) (B,A,C) (A,E,7)T(A,T,B) (A,U,2,) (A,W,3) LØC5

which says that *if any* of the following is true

that scanner *B* is *R*ight of x = *10*, or
that scanner *B* is *A*bove (in a line higher than) scanner *C*, or
that scanner *A* is sitting on a number *E*qual to *7*

*T*hen the following operations are performed

scanner *A* moves *T*o the same surface and the same square as scanner *B*,
scanner *A* moves *U*p *2* squares, and
scanner *A* *W*rites the number *3*, and

control then goes to the line of coding labeled LOC5. If none of the three elementary conditions is satisfied, no operations are performed and control goes to the next line of coding.

The scanner language permits a large number of different elementary tests on the positions of scanners and on the numbers on which they sit. There are also a large number of elementary operations for moving scanners and for changing the numbers on which they sit. A complete list of these basic tests and operations, and the flexible formats in which they may be used, is given in Appendix A. A concise summary of the scanner language appears there in Tables A.1 and A.2.

The other part of the movie language may be called the "movie language proper." It consists of more powerful instructions which in general compile into calls to subroutines written primarily in the scanner language. (These subroutines actually use an additional set of 26 scanners which the programmer need not know about.) Instructions of the movie language proper fall logically into three categories: instructions for controlling the output or temporary storage of pictures, instructions for performing drafting and typing operations, and instructions for modifying the contents of rectangular areas. These categories will be discussed briefly in turn.

The outputting of movie frames (i.e., the writing of the tape which will control the S-C 4020) is performed by an internal "camera" subroutine. The subroutine has many modes of operation, which are determined by "camera settings." One setting "aims" the camera at all of or part of a surface; another specifies how many frames of the *final* film are to be produced for every frame of the master film. A third setting specifies the transliteration from digits 0 through 7 to the typeable characters of the 4020 charactron. Other settings specify what output the camera routine should produce on the normal printer, for monitoring purposes. Facilities are also available for temporarily storing entire contents of surfaces on a disc file. Up to 440 complete fine-resolution movie frames may be stored on the disc at any one time.

The facilities for drafting include the ability to draw straight lines, arcs, and arbitrary curves. These lines may be made to appear instantaneously in the movie, or they may be

made to appear gradually as they are drawn. Lettering may be done by "typing" letters of six different sizes, the smallest letters each covering an area 4 squares wide and 5 high, the largest covering an area 15 squares wide and 21 high.

Finally, the entire contents of a surface or a rectangular subarea may be changed in many ways. The area may be "painted" by filling all of its squares with one particular digit, or another area of similar size and shape may be copied into it. The contents of the area may be shifted an integral number of positions up, down, right, or left, or they may be "expanded" (by repeating rows or columns) or "squashed" (by deleting rows or columns) in any of the four directions. Certain local operations may be performed throughout the area, such as adding incremental layers to regions defined by a certain number, or rounding sharp corners. There are operations which approximate the effect of a zoom lens by enlarging or reducing by an integral factor the pattern of numbers within a rectangular area while the camera routine is putting out pictures of the intermediate states of the picture. There is also a facility for "dissolving" one picture onto another by sprinkling the new numbers onto the old ones, again while the camera routine is outputting the intermediate states of the dissolve.

The movie language proper is described in detail in Appendix B. A programming example, involving instructions both of the scanner language and of the movie language proper is presented in Appendix C. This sample program was actually the program used to produce Fig. 2a of this paper.

Uses of the Movie System

This movie language may be used to produce many types of simple animated movies. It may be used, for example, to produce visual displays for psychophysical experiments, or to produce a more common type of movie such as the expository educational film.

The system may also be used to convert the output of computer-performed experiments into visual displays. For example, the person experimenting with heuristics for automatic layout of printed circuits may wish to watch in a movie the computer's attempts to search efficiently for wire paths.

Costs for producing movies by this means are low, and movies can also be produced quickly, primarily because only a few people are involved. Simple educational movies cost a few hundred dollars per minute, with the cost split approximately equally three ways: programmer's time and overhead, computer time, and standard movie laboratory operations.

Much of the power of the movie system as implemented derives from the fact that it has been constructed entirely within the framework of MACRØ FAP. The instructions of the scanner language and of the movie language proper are actually macro-instructions. As such, they may be interspersed with instructions of the basic FAP language or, more important, with higher-order macro-instructions which the programmer has defined in terms of the original movie instructions. Appendix C contains examples of such higher-order macros which were developed, in some cases because they were more powerful, and in others because they more exactly matched the requirements of a specific job.

Acknowledgements

I wish to thank my associates at Bell Telephone Laboratories who have helped and encouraged me in this work. Particular thanks go to Paul W. Hoff, who wrote and checked out many of the subroutines.

REFERENCES

1. S-C 4020 High Speed Microfilm Recorder, Product Specification 281001-241A, September 8, 1960, Stromberg-Carlson, San Diego, California.
2. 7090 Bell Telephone Laboratories Programmer's Manual, Bell Telephone Laboratories, Incorporated, Murray Hill, New Jersey.

APPENDIX A

The Scanner Language

The scanner language is that part of the movie language by which the programmer explicitly performs tests and operations on the 26

scanners—those reading and writing heads illustrated in Fig. 3 which scan and operate on two-dimensional arrays of numbers. This appendix describes the formats and uses of scanner language instructions; Appendix B describes the more powerful movie instructions which constitute the "movie language proper." An example of programming with both kinds of instructions is presented in Appendix C.

Scanner Initialization

A scanner, before being used, must be initialized by an instruction with the special format:

PLACE sc,surf,x,y
(e.g. PLACE D,BB,92,5)

which has the effect of placing scanner *sc* on the surface *surf* at coordinates *x,y*. The scanner may be any one of the 26 available, A,B,. . .,Z, and the surface may be any one of those illustrated in Figs. 2a. and 2b. The coordinates must refer to a square which is actually on the surface: the programmer should note that the bottom left square coordinates (0,0) and the top right square of, say, a 252 × 184 surface, has coordinates (251, 183).

General Instruction Formats

Scanner instructions are generally expressed in terms of *elementary tests* on positions of scanners and the numbers in the squares they are sitting on, and in terms of *elementary operations* directing scanners to move or write new numbers into these squares.

Instructions—or lines of coding—are of two basic types: unconditional and conditional. The *unconditional* instruction may be illustrated schematically as

Symb	THEN	( )( ) . . . ( )	Symb2
symbol (Optional)		list of elementary operations	goto (Optional)

and it contains the following parts:

1. In the location field of the card, an optional FAP symbol which is *not* a single letter or double letter (the symbols A,AA, B,BB, . . . Z,ZZ have been pre-empted by the movie system).
2. In the operation field of the card, the macro name THEN.
3. In the variable field of the card, a list of operations, followed by an optional single symbol indicating where control is to go after the operations have been performed. If there is no goto, control passes to the next line of coding. If there is a goto, the entire list of operations may be missing. The length of the list is limited by the restriction that the operation and goto must appear on one card from columns 16 through at most column 72.

The *conditional* instruction is similar in format except that preceding the list of operations there appears a list of elementary conditions followed by the delimiter 'T', and that the name of the macro-instruction indicates which of four logical functions on the conditions must be satisfied in order that the operations be performed:

Symb	IFxxx	( )( ) . . . ( )	T	( )( ) . . . ( )	Symb2
Symbol (Optional)		list of elementary conditions		list of elementary operations	goto

If the list of operations is null, the 'T' is omitted and a goto must appear. Again, the length of the lists is limited only by the requirement that the list of conditions must start in column 16 and the instruction may extend at most through column 72 of the same card.

Each of the 4 MACRØ names requires satisfaction of a different logical function of the conditions, as follows:

IFANY, satisfied if any of the elementary conditions is satisfied.

IFALL, satisfied if all of the elementary conditions are satisfied.

IFNØNE, satisfied if none of the elementary conditions are satisfied.

IFNALL, satisfied if not all of the elementary conditions are satisfied.

In each case, if the compound condition is satisfied, then the indicated operations are performed and if there is a goto, control goes to the indicated point in the program, otherwise to the next line of coding. If the compound condition is not satisfied, no operation is performed and control goes to the following line of coding.

In addition to the five basic macro-instructions, the following synonyms are built into the system:

ANY,	synonym	for	IFANY
ALL,	"	"	IFALL
NØNE,	"	"	IFNØNE
NALL,	"	"	IFNALL
IF,	"	"	IFALL
NØT,	"	"	IFNØNE
EITHER,	"	"	IFANY
BØTH,	"	"	IFALL
ELSE,	"	"	THEN

The first four of these are simply abbreviated notations to facilitate programming. The others enable the program to be more easily read and understood. IF and NØT are suggested in place of IFALL and IFNØNE, respectively, where there is just one elementary condition. Likewise, EITHER and BØTH are suggested in place of IFANY and IFALL, respectively, where there are two elementary conditions. ELSE is suggested in place of THEN when it follows a conditional instruction that has a goto.

Elementary Conditions or Tests

An elementary condition is a simple test performed on the position of a scanner or on the number this scanner is sitting on. Every elementary condition is written as a triplet of arguments separated by commas and delimited by parentheses. It has the form

(scnr,rel,quant)

where *scnr* is a single-letter name of a scanner, *rel* is a single letter designating a particular relation, and *quant* specifies either directly or indirectly, the quantity or coordinate involved

TABLE A.1

Elementary Scanner Conditions and Their Formats

Key:	α,β	scanner: A,B,C,... Z
	n(α)	the number α is sitting on
	K (n(α))	bit-by-bit complement of n(α)*
	X(α)	abscissa of α: 0,1,2...
	Y(α)	ordinate of α: 0,1,2...
	n	a decimal number (if n is a number in a square, $0 \leq n \leq 7$)

"Quantity" is a scanner name		*"Quantity" is a number*	
1. Tests on position of scanner α. Is it true that α is:			
at X (β)	(α,X,β)	at $X = n$	(α,X,n)
at Y (β)	(α,Y,β)	at $Y = n$	(α,Y,n)
to the *R*ight of X(β)	(α,R,β)	to the *R*ight of $X = n$	(α,R,n)
to the *L*eft of X(β)	(α,L,β)	to the *L*eft of $X = n$	(α,L,n)
*A*bove Y(β)	(α,A,β)	*A*bove $Y = n$	(α,A,n)
*B*elow Y(β)	(α,B,β)	*B*elow $Y = n$	(α,B,n)
2. Tests on the number scanner α is sitting on. Is it true that n(α)			
is *E*qual to n(β)	(α,E,β)	is *E*qual to the number n	(α,E,n)
is *N*ot equal to n(β)	(α,N,β)	is *N*ot equal to n	(α,N,n)
is *S*maller than n(β)	(α,S,β)	is *S*maller than n	(α,S,n)
is *G*reater than n(β)	(α,G,β)	is *G*reater than n	(α,G,n)
contains all *Z*ero bits of n(β)*	(α,Z,β)	contains all *Z*ero bits of n*	(α,Z,n)
contains all *O*ne-bits of n(β)*	(α,Ø,β)	contains all *O*ne-bits of n*	(α,Ø,n)

* With numbers expressed in binary notation.

in the test. If *quant* is a number it specifies the quantity directly; if *quant* is a letter than it specifies the scanner whose number or position is involved in the test. A complete list of tests and the letter by which they are designated appears in Table A.1.

Elementary Operations

An elementary operation, like an elementary condition, is written as a triplet of arguments, separated by commas and delimited by parentheses, and has the form

(scnr,op,quant)

Here *scnr* is the single-letter name of the scanner which performs the operation, *op* is a single letter designating the operation to be performed, and *quant* generally specifies directly or indirectly a quantity involved in the operation: as in elementary tests, a number specifies the quantity directly, whereas a letter specifies the scanner whose number or position is to be used. One exception is the operation

(α,Z,β)

which specifies that both scanners α and β are to exchange numbers, i.e., each writes the number that the other was just sitting on.

TABLE A.2

Elementary Scanner Operations and Their Formats

(Key: same as for Table A.1)

"Quantity" is a scanner name		*"Quantity" is a number*	
1. Operations for moving α:			
To surface and position of β	(α,T,β)		
horizontally to $X(\beta)$	(α,X,β)	horizontally to $X = n$	(α,X,n)
vertically to $Y(\beta)$	(α,Y,β)	vertically to $Y = n$	(α,Y,n)
		Up n squares	(α,U,n)
		Down n squares	(α,D,n)
		Right n squares	(α,R,n)
		Left n squares	(α,L,n)
Move one square according to $n(\beta)$ *	(α,M,β)		
2. Operations for changing the number α is sitting on, by:			
Writing the number $n(\beta)$	(α,W,β)	Writing the number n	(α,W,n)
Writing K $(n(\beta))$	(α,K,β)		
exchanging $n(\alpha)$ and $n(\beta)$	(α,Z,β)		
bit-by-bit ϕRing by $n(\beta)$	(α,O,β)	bit-by-bit ϕRing n	(α,ϕ,n)
bit-by-bit ANDing by $n(\beta)$	(α,A,β)	bit-by-bit ANDing n	(α,A,n)
adding $n(\beta)$ **	(α,E,β)	adding n**	(α,E,n)
subtracting $n(\beta)$ **	(α,F,β)	subtracting n**	(α,F,n)
multiplying by $n(\beta)$ **	(α,G,β)	multiplying by n**	(α,G,n)
dividing by $n(\beta)$ **	(α,J,β)	dividing by n**	(α,J,n)
Setting α's memory to $n(\beta)$	(α,S,β)	Setting α's memory to n	(α,S,n)

* Step up if $n(\beta) = 4$, step right if $n(\beta) = 5$, step down if $n(\beta) = 6$, step left if $n(\beta) = 7$, otherwise no motion.
** Result reduced modulo 8.

A list of elementary operations appears in Table A.2. It should be noted that certain operations require that the quantity involved always be indicated directly as a number, whereas certain other operations require the quantity to be specified indirectly by the name of a scanner.

It should also be noted that there is no particular relation between the interpretation of a specific triplet as a test and the interpretation of the same triplet as an operation. For example the triplet

(B,A,6)

interpreted as a test means "Is scanner *B* above y = *6*?", whereas as an operation it means "The number that scanner *B* is sitting on should have *A*NDed onto it the number *6* (i.e., its low order bit should be forced to zero)." Whether a triplet is to be interpreted as a test or an operation is determined by its position in the line of coding.

The available operations permit scanners to be moved beyond the limits of their surfaces as defined by Figs. 2a and 2b; special consideration should be given to the results of such operations. The surfaces of Fig. 2a act as helices, such that a single step "right" from the right column of the surface places the scanner on the *leftmost* column of that surface but *one row below* where it started. Conversely, a step left over the edge places it on the right edge one row *above* where it started. Motion above and below the top and bottom edges is legal, but the scanners will perform a "no-operation-and-continue" instead of altering any "numbers" that they might be sitting on there. Furthermore, the programmer should be aware that the y-coordinate is treated modulo 2^{15}, so that after a step down from y = 0, a test on its position will result as if the scanner were at y = 32,767.

The connectivity of the surfaces of Fig. 2b is similar to that of surfaces in Fig. 2a except that after stepping over the right edge of the surface, a scanner enters a no-man's-land a few squares wide; after successive steps through this region, the scanner appears, as in the case of the other surfaces, on the *left* edge, one row below the one it started on.

Another precaution regarding the use of the operations of Table A.2 concerns the case in which two or more scanners are sitting on the same square, as they would be, for example, after an operation of the form

$$(\alpha,T,\beta)$$

If one of the scanners changes the number on the square in any way, the other scanners on that same square do not "know" that the number has been altered, and subsequent tests or operations involving the number under one of the other scanners may yield an erroneous result. In general a scanner updates its memory of the number it is sitting on only when it moves to a square or when it itself changes this number. "Moving" a scanner α to its own position by the operation

$$(\alpha,T,\alpha)$$

will always properly update α's conception of the number it is sitting on.

Finally, a scanner's memory may be deliberately set to a number which has no relation to the number it is sitting on by the last operation of Table A.2:

$$(\alpha,S,\beta) \text{ or } (\alpha,S,n)$$

Subsequently, and until another operation is performed with scanner α, all tests and operations involving α result as if α were sitting on $n(\beta)$ or on n. This provides a useful way of passing information (numbers) to subroutines without actually having to write these numbers on a surface.

Subroutines

In order to perform a subroutine beginning at symbolic location *sub*, the special triplet

$$(QQ,P,sub)$$

is inserted as an ordinary triplet in a list of elementary operations. This special operation saves on a pushdown list the location at which the program was operating, and it transfers control to the subroutine.

Exit from a subroutine is accomplished by use of the special goto which is identically

$$QQ$$

Since subroutine returns are recorded on a pushdown list, a subroutine may use itself, provided that the programmer has in some way prevented indefinite recursion into the routine.

Double-letter "scanners"

In addition to the 26 scanners A through Z, there is a special double-letter scanner sitting on the upper right hand corner of each surface. The name of each such special "scanner" is identical with the name of the surface. These scanners *may not be moved* but they may be used to write numbers on their particular squares. Their names may also be used as the "quantities" in tests and operations on regular scanners. For example, there is a scanner

BB sitting on the upper right hand corner of surface BB at (251,183) and the instruction

IF (A,X,BB)T(A,T,BB)

would function the same as the instruction

IF (A,X,251)T(A,Y,183)

The double-letter scanners may also be used on the same basis as single-letter scanners for specifying areas to which the higher order operations of Appendix B are to be applied.

APPENDIX B. THE MOVIE LANGUAGE PROPER

In addition to the scanner instructions of Appendix A, the movie programmer may use the more powerful instructions of the "movie language proper," described below. These are, in general, macro-instructions which compile into calls to subroutines which themselves are written mostly in the scanner language.

The movie instructions fall naturally into four categories, including instructions for

(1) controlling output of pictures and temporarily storing pictures and retrieving them from the disc file,
(2) performing drafting and typing operations,
(3) performing "instantaneous" operations on the contents of rectangular area or surfaces, and
(4) performing "dynamic" operations on the contents of rectangular areas or surfaces.

An *instantaneous* operation is one which is performed and completed between output of adjacent frames of film, whereas a *dynamic* operation is one which is performed gradually while several frames of pictures are being output by the "camera" output routine.

These four groups of macro-instructions will be discussed in turn. The format of each instruction will be illustrated and described in terms of dummy arguments and in most instances an example of the use of the instruction will be given. A résumé of all macro-instruction formats is given in Table B.1, which also contains a list of the more common dummy arguments used to describe these instructions.

1. *Instructions for Output and Temporary Storage*

An output routine or "camera" within the 7090 is used to write information on the magnetic tape which is later used to direct the S-C 4020 in exposing film. The camera routine is initiated by the instruction

CAMERA n (n optional)
(e.g. CAMERA 3)

where n is the intended number of identical frames to be produced in the final film. Only one frame is produced by the 4020, with the number n printed just above this frame. If n is not specified in the CAMERA call, then the number used is that last specified by the setting

FRAMES n
(e.g. FRAMES 2)

This setting is useful for controlling the apparent speed at which dynamic operations are performed, since the subroutines of the system which perform dynamic operations contain CAMERA calls without specification of n. In the event that the specified or effective n is zero, the camera call is ineffective, and no picture is output.

Besides the FRAMES setting, there are several other settings which control the operation of the camera routine. Camera settings, to be discussed in turn, include

FRAMES	(how many identical frames in final movie?)
AIM	(what surface area to output?)
FINE or CØARSE	(what resolution?)
FILTER	(what transliteration during output?)
SAMPLE	(how often to monitor results on printer?)
LINES	(which lines of picture to print when monitoring?)
FILM or NØFILM	(film output, or just monitor output?)

TABLE B.1

Résumé of Movie Macro-instructions

Key	(see text for meanings of dummy arguments not listed in the key)
scTR	scanner sitting on *T*op *R*ight corner of affected rectangular area
scBL	scanner sitting on *B*ottom *L*eft corner of affected rectangular area. If scBL = '0' then (0,0) of scTR's surface is implied
sc,sc1,sc2, . . .	scanner names
surf	name of a surface
mode	stands for one of the following arguments designating the three different ways of changing numbers on the surface: WRITE (replace old number by new one) AND (bit-by-bit logically AND old and new numbers) ØR (bit-by-bit logically ØR old and new numbers)
ns,ns1, ns2, . . .	indicate *n*umbers to be used or the names of *s*canners sitting on the numbers to be used
n,n1,n2, . . .	numbers
width	a number from 1 to 6 designating the width in squares of a line, arc, curve, or border to be drawn
dir	stands for one of the four directions: UP, RIGHT, DOWN, or LEFT
orient	stands for one of the eight basic reorientations: ST standard orientation 90R rotated 90° right (clockwise) 90L rotated 90° left 180 rotated 180° X reflected through x axis Y reflected through y axis YEX reflected through line y = x YEMX reflected through line y = –x
amt	the number of squares of shift, rotation, etc.
fctr	an integer from 2 to 6 specifying the factor of magnification, reduction, "stretch" or "press"
speed	a number specifying the "speed" at which a line is drawn, i.e., the number of squares it advances between successive calls of CAMERA (if this number is very large, the entire line appears instantaneously)

1. *Instructions for output and temporary storage*

CAMERA	n	(*n* optional) or CAMERA UNTIL,n
FRAMES	n	($0 \leq n$)
FILTER	n	($0 \leq n \leq 10$)
TABLE	n,n0,n1,n2,n3,n4,n5,n6,n7	
AIM	sc	
SAMPLE	n	
LINES	n	(*n* is 11 octal digits)

FILM	
NØFILM	
UNTIL	n,goto
RESET	n
CØARSE	
FINE	
STØRE	surf,where (*where* is NEXT,PREV, a number, or missing)
RETREV	surf,where (*where* is NEXT,PREV, a number, or missing)

2. Instructions for drafting and typing

(The entire scanner language of Appendix A may be considered to be in this category)

	LINE	sc1,sc2,mode,ns,width,speed
	ARC	sc1,center,d,mode,ns,width,speed,t1,q1,t2,q2 (t2,q2 optional)
	TRACE	symb1,length,sc,orient,mode,ns,width,speed
	TYPE	symb2,sc,size,Hspace,Vspace,mode,ns
	⋮	
symb1	ØCT	n, (specification of curve to be drawn by TRACE)
symb2	BCI	n, (specification of text to be typed by TYPE)

3. Instructions for instantaneous operations on rectangular areas

PAINT	scTR,scBL,mode,ns
BORDER	scTR,scBL,width, mode,ns
SHIFT	scTR,scBL,dir,amount
RØTATE	scTR,scBL,dir,amount,n (n optional)
EXPAND	scTR,scBL,dir,rep1,rep2
SQUASH	scTR,scBL,dir,del,kp
CØPY	scTR,scBL,mode,orient,sc3,sc4,n (n optional)
CENTER	scTR,scBL
GRØW	scTR,scBL,ns1,ns2,ns3,goto (goto optional)
SMØØTH	scTR,scBL

4. Instructions for dynamic operations on rectangular areas

DISØLV	scTR,scBL,sc3,pat
ZØØMIN	scTR,scBL,fctr
REDUCE	scTR,scBL,fctr
STRECH	scTR,scBL,dir,fctr
PRESS	scTR,scBL,dir,fctr

These settings must be made before the first CAMERA call, but they may be changed at any later point in the program.

The surface area to be output is determined by the position of that scanner *sc* specified in the last previous setting of the form

	AIM	sc
(e.g.	AIM	BB)
(or	AIM	A)

The top right corner of the picture which is output by a CAMERA call is the current position of that scanner specified by the AIM setting. If the scanner moves, the camera tracks the scanner.

The output mode (coarse or fine) is determined by which of the settings,

	FINE
or	CØARSE

occurred last. The setting FINE specifies that henceforth and until encountering the next COARSE, the area to be output is a rectangular array of squares 252 wide and 184 high. The setting CØARSE specifies that henceforth and until the next FINE, the area to be output is

an array 126 wide and 92 high. In this case, the spots are displayed at twice the spacing for FINE output in order to fill the complete movie frame.

It will normally be the case that the picture on film is to be composed of other charactron characters than the digits 0 through 7. This requires transliteration during output, specified by the setting

```
        FILTER n        (0 ≤ n ≤ 10)
(e.g.   FILTER 5)
```

where n is either 0, specifying no transliteration, or a number from 1 to 10 specifying one of ten available transliteration tables. Each table thus used may be set up or later changed by a command

```
        TABLE  n,n0,n1,n2,n3,n4,n5,n6,n7
(e.g.   TABLE  5,60,15,14,72,13,16,54,53)
```

which causes table n to transliterate 0 into *n0*, 1 into *n1*, 2 into *n2*, etc. The characters which may be used for output on the charactron are any of the sixty-four octal characters 0 through $(77)_8$ except $(12)_8$, $(52)_8$ and $(56)_8$. The specific example above gives a recommended grey scale, transliterating 0 into blank $(60)_8$, 1 into apostrophe $(15)_8$, 2 into quote (14), 3 to degree sign $(72)_8$, 4 to equal sign $(13)_8$, 5 to delta $(16)_8$, 6 to asterisk $(54)_8$, and 7 to dollar sign $(53)_8$ which is the darkest typeable character on the charactron.

In addition to film output, printed output may be produced for monitoring purposes. The setting

```
        SAMPLE n
(e.g.   SAMPLE 24)
```

says that printed output is to be produced for every n^{th} frame of the final movie, except that this output will be produced at most once in any one CAMERA call. Thus if the sampling rate is 24 and the instruction, CAMERA 150, is encountered, only one printed output is produced during this operation, labeled with the beginning and ending frame numbers of the corresponding sequence of identical frames in the final movie. The part of the frame which is to appear in the printed output is predetermined by the bit pattern of the 11-digit octal number, n in the setting,

```
        LINES   n
(e.g.   LINES   1400000003)
```

The first octal digit is 1 or 0, stating that lines x = 91 and x = 90 of a CØARSE picture should or should not be printed, each successive *bit* states whether or not the next *3* lines should be printed. In the example given, only the top 5 lines and the bottom 6 lines would be printed. If the output mode is FINE, then only the odd columns of the corresponding odd-numbered rows are printed. If the above sample LINES setting were used for fine-resolution output, then odd-numbered positions of rows 183,181, 179,177,175,11,9,7,5,3 and 1 would be printed (counting the bottom line as line zero).

The actual production of film is enabled or disabled by the instructions

```
        FILM
or      NØFILM
```

each of which compiles into a single machine instruction. Common practice is to begin every program with this sequence of instructions, and to test the program in this form, producing only the printed output for monitoring purposes. When the program has been checked out, a production run is performed in which a correction card replaces the NØFILM instruction with a NØP machine instruction.

The system contains a counter which counts frames of the final movie. This counter may be interrogated, and flow of control directed by the branch

```
        UNTIL   n,goto
(e.g.   UNTIL   2400,AGAIN)
```

which causes control to go to the indicated *goto* if the current frame count is below the specified number n. A special format for the CAMERA call also uses the frame counter

```
        CAMERA  UNTIL,n
(e.g.   CAMERA  UNTIL,2496)
```

This call directs the camera routine to produce one frame, as does a normal call, but in this case the number in the frame line—specifying how many times this frame is to be repeated in the final movie—is made just large enough to bring the frame count up to the specified n. The frame counter may be reset by the program to any n by the command

```
        RESET   n
(e.g.   RESET   4800)
```

Operations and tests involving the frame counter are intended primarily to facilitate synchronization of the movie with a sound track which is added later by traditional methods.

In addition to putting out pictures on film, contents of entire surfaces may be temporarily stored on and retrieved from the disc file. There are 440 available storage areas on the disc, numbered 1 through 440, each capable of storing the entire contents of any surface except surfaces VV or CC. Storing of surface *surf* is accomplished by the instruction

```
        STØRE   surf,where (where is
                n,NEXT,PREV, or
                null)
(e.g.   STØRE   AA,150)
```

and the picture is retrieved by the instruction

```
        RETREV  surf,where (where is
                n,NEXT,PREV, or
                null)
(e.g.   RETREV  AA,PREV)
```

In either case, the storage area *where* may be specified explicitly by a number, or implicitly by three other possibilities: if *where* is missing in the call, the last area used in a STØRE or RETREV command is used; NEXT implies the next higher area than the last one used, and PREV implies the next lower area than the last one used. The surfaces VV or CC *can* be effectively stored by storing the contents of both AA and BB, since the latter occupy the same internal 7090 storage space. For example, contents of surfaces VV (or CC) may be stored in areas 150 and 151 by the sequence

```
STØRE   AA,150
STØRE   BB,NEXT
```

and they may then be retrieved by the sequence

```
RETREV  BB
RETREV  AA,PREV
```

One precaution must be taken after retrieval of a picture: a scanner a now sitting on this surface may not act as if it were on the corresponding new number until it is first "moved" by some such scanner instruction as (a,T,a) or (a,X,a) or (a,Y,a).

2. Instructions for Drafting and Typing

The instructions for drafting have the names LINE, ARC, and TRACE; the one instruction for typing is called TYPE. These are all dynamic operations in the sense that while each is being executed it is interrupted periodically to allow the camera routine to output pictures. For the TYPE instruction, the interruption occurs after every large character (made up of a rectangular matrix of numbers) has been typed, and the camera routine may be rendered ineffective by a previous setting 'FRAMES 0'. For the drafting operations, the interruption occurs every time the line being drawn has advanced another n squares, where n is specified in the instruction as the "speed" at which the line is to be drawn. In this case, the interruption is avoided, or it is rendered ineffective, by a very high speed (e.g. 5000) or by a previous setting 'FRAMES 0'.

An approximation to a straight line is drawn from scanner *sc1* to *sc2* by the instruction

```
        LINE    sc1,sc2,mode,ns,width,
                speed
(e.g.   LINE    A,B,WRITE,5,3,9)
```

The line is drawn by using the number *ns* (or the number that *ns* is sitting on if *ns* is a scanner name): if *mode* is WRITE, this number is used to replace the numbers on affected squares, whereas if *mode* is ØR or AND, the new number is ØRed or ANDed, bit by bit, with the previous number in each affected square. The line is drawn by an imaginary stylus which moves by stepping either horizontally or vertically to the adjacent square which lies closest to the ideally straight line. On each square thus traversed, an approximately circular dot of diameter *width* is centered, and a picture is produced after advancing each n squares along the line, where n is the desired *speed*.

An arc is drawn in a manner similar to the drawing of a line, by the instruction

```
                                    optional
        ARC     sc1,sccent,d,mode,ns,width,speed,t1,q1,t2,q2
(e.g.   ARC     A,B,CCW,ØR,6,5,12,Y,B,R,B)
```

where the arguments *mode,ns,width,* and *speed* have the same meaning as for LINE. The arc begins at the position of scanner *sc1*, and proceeds with scanner *sccent* as its center, in direction *d*, which is CW for clockwise or CCW for counterclockwise. The arc is terminated when the drawing stylus, which itself may be thought of as a scanner, satisfies the joint condition, expressed in scanner language,

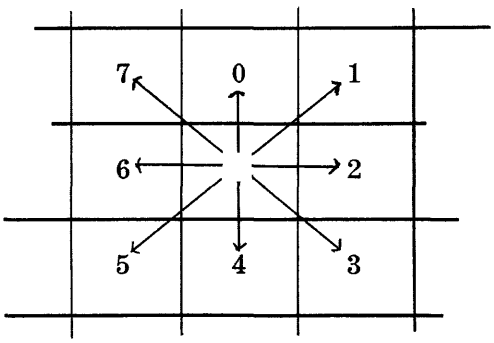
(stylus,*t1,q1*) (stylus,*t2,q2*)

or when it satisfies simply the first condition if the second condition is not given in the ARC instruction. If the terminating condition is not satisfied within 1000 elementary steps, the program stops. In the specific example above, an arc of width 5 is initiated at scanner A and is drawn about scannerB as center in a counterclockwise direction at speed of 12, by ØRing the number 6 onto affected squares. The arc is terminated when the drawing stylus is at the same height as scanner B and right of scanner B.

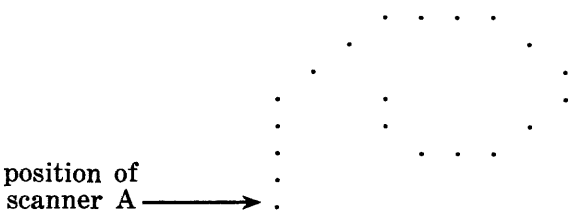
A curve of arbitrary shape may be traced by an instruction of the form

```
          TRACE    Symb,length,sc,orient,mode,ns,width,speed
e.g.      TRACE    CURVE7,19,A,ST,WRITE,2,1,10
            ⋮
CURVE7    ØCT      000011122233,455667000000
```

where *mode,ns,width* and *speed* have the same meanings as for LINE. The argument *length* is the length of the curve to be drawn, expressed in elementary steps from one square to the next, *sc* is the scanner at which the curve is to start, and *orient* is one of the eight possible reorientations of the basic curve (see the Key in Table B.1). *Symb* is a FAP symbol indicating the location at which a description of the basic curve is given in terms of a sequence of incremental steps. Each step is here specified by one of eight octal digits which stand for the eight possible directions for these steps:



In the specific example above, if the original surface contained only zeros, and if on output zeros are transliterated to blanks and 2's to dots than the result would be



If the same curve (with the same description) had been drawn in the YEMX orientation (reflected through the line y = — x) it would have started with a straight section going left and then would have spiralled counterclockwise.

The operation of "typing" in the movie language is done by affecting appropriate patterns of squares. The general form of the typing instruction is

```
          TYPE     symb,sc,size,Hspace,Vspace,mode,ns
e.g.      TYPE     TEXT5,B,5x7,1,3,WRITE,6
            ⋮
TEXT5     BCI      3,*(        BELL
          BCI      3,*/   *TELEPHØNE
          BCI      3,*/*LABORATØRIES,
          BCI      3,*/        *INC.        *.
```

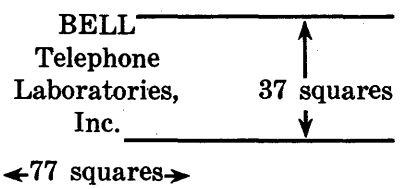
where *symb* is a FAP symbol identifying the description of the text to be typed, *sc* is the scanner specifying the position of the bottom left hand corner of the first character to be type, and *size* is one of the following sizes of characters which may be typed:

4x5	(4 squares wide, 5 high)
5x7	(5 squares wide, 7 high)
7x10	(7 squares wide, 10 high)
8x11	(8 squares wide, 11 high)
10x14	(10 squares wide, 14 high)
15x21	(15 squares wide, 21 high)

The argument *Hspace* specifies the horizontal spacing in squares between the characters of a line, whereas *Vspace* specifies the number of squares of vertical spacing between successive lines. The *mode* may be AND, ØR, or WRITE as with the drafting operations, but the programmer must note that only those squares corresponding to the body of the letter are affected: background areas, such as the center of a '0' or the entire area of a blank are unchanged regardless of the mode used. The argument *ns* is either the number to be used in changing affected squares, or the name of scanner sitting on the number to be used. The description of text to be typed is written out on FAP BCI cards. All of the 48 standard IBM Hollerith characters may be typed. The following special sequences, all starting with '*', have the indicated meanings for the typing routine:

*.	stop typing
*(	shift to upper case (affects typing of letters only)
*)	shift to lower case (affects typing of letters only)
*L	where L is any letter: Type this letter in upper case, then shift to lower case
*	return carriage: Return to starting point of previous line, then go down letter height plus vertical spacing
**	type the character '*'

In the specific case of the example illustrated, the result would appear approximately as follows:



Other examples of typing operations appear in Appendix C.

3. Instructions for Instantaneous Operations on Rectangular Areas

The contents of rectangular areas may be altered by any of a large number of instantaneous operations—operations which appear to be performed instantaneously in the movie because their respective subroutines contain no CAMERA calls. The formats and uses of these instructions are described in the following paragraphs. In all cases, the rectangular areas to be changed are specified by the positions of scanners. The dummy argument "scTR" in a format statement stands for "*sc*anner defining the *T*op *R*ight corner of the area." The dummy argument "scBL" means "either the name of a *sc*anner defining the *B*ottom *L*eft corner of the area or else '0', meaning x = 0, y = 0 of the surface that scanner scTR is on."

A rectangular area may be "painted"—every square changed by using the same number—by the instruction

```
        PAINT   scTR,scBL,mode,ns
(e.g.   PAINT   ZZ,0,WRITE,0)
```

where *mode*, as before, is AND, ØR or WRITE, indicating whether the change is to be accomplished by ANDing, ØRing, or replacement, respectively and *ns* is either the number to be used or the name of a scanner sitting on the number to be used. In the example above, the entire surface ZZ is "cleared" to zeros.

A rectangular border of any thickness may be produced just within the periphery of the rectangular area by the instruction

```
        BØRDER  scTR,scBL,width,mode,ns
(e.g.   BØRDER  A,B,9,ØR,A)
```

where *width* is the thickness of the desired border, *mode* and *ns* indicate the manner in which numbers are to be changed and the number to be used, as in the case of PAINT.

The contents of a rectangular area may be shifted up, right, down or left any number of positions—up to the dimension of the area in this direction—by the instruction

SHIFT scTR,scBL,dir,amt
(e.g. SHIFT AA,0,UP,36)

where *amt* is the number of squares of shift and *dir* is the direction of shift, UP,RIGHT, DØWN, or LEFT. As a result of the shift operation, material is lost at one edge of the rectangular area and the "vacated" area is filled by repeating the original row or column just within the edge that contents of the area were shifted away from.

The material within a rectangular area may be "rotated," as if around a cylinder, by the instruction

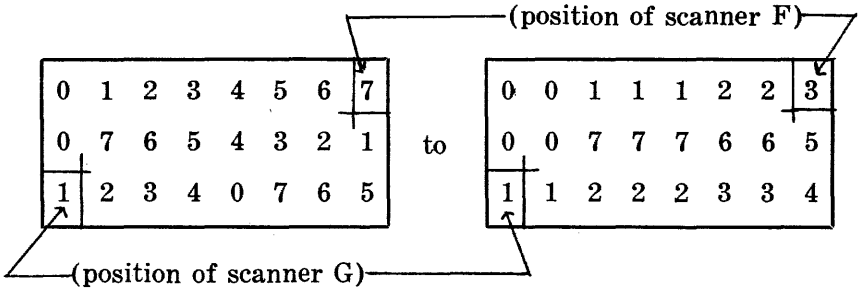
RØTATE scTR,scBL,dir,amt,n (n optional)
(e.g. RØTATE C,D,LEFT,5)

The effect is the same as for SHIFT except that the material which is lost at one edge is introduced at the opposite edge. Furthermore, if n is given, the material which "goes around the back" to be reintroduced is transliterated accoring to transliteration table n. This table must have been previously established by a TABLE instruction such as is used to specify transliteration during output, but only the three lowest-order bits of the new numbers can be used for the RØTATE transliteration.

The pattern of numbers within a rectangular area may be "expanded" upward or downward by duplicating certain rows, or it may be expanded to the right or left by duplicating certain columns. The instruction

EXPAND scTR,scBL,dir,rep1,rep2
(e.g. EXPAND F,G,RIGHT,2,3)

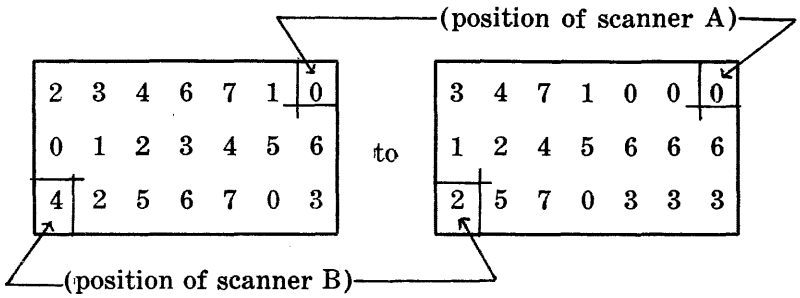
causes the material to be expanded in the direction *dir* (UP, RIGHT, DØWN, or LEFT) by starting at the edge row or edge column opposite the direction of expansion and repeating the first row or column *rep1* times, the next row or column *rep2* times, the next *rep1* times, etc., until the entire rectangular area has been refilled. The sample EXPAND instruction above would change the pattern



The pattern of numbers in a rectangular area may also be "squashed" toward one edge by depleting certain rows or columns and by repacking the remaining numbers. The instruction

SQUASH scTR,scBL,dir,del,kp
(e.g. SQUASH A,B,LEFT,1,2)

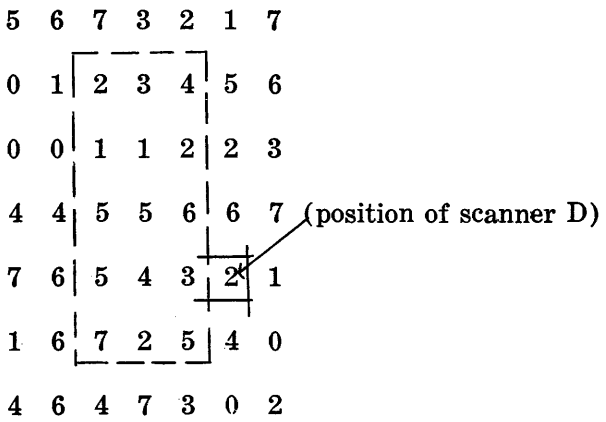
performs such an operation by starting at the edge specified by the direction *dir* and alternately deleting *del* columns (rows) and keeping *kp* columns (rows). The remaining columns (rows) are closely packed and the vacated area is filled by duplicating the original column (row) just within the edge from which the motion has occurred. The sample instruction above would change the pattern



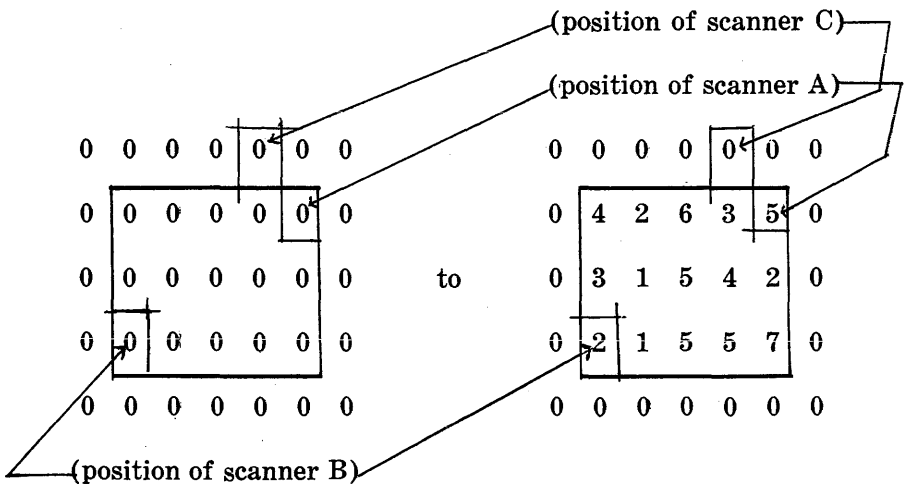
A rectangular area may be filled or changed by "copying" from another area. This is accomplished by the powerful and versatile instruction

CϕPY scTR,scBL,mode,orient,sc3,sc4,n
(n optional)
(e.g. CϕPY A,B,WRITE,90L,C,D)

Here, as before, *scTR* and *scBL* define the affected area, *sc3* is a scanner on this same surface (it may in fact be either scTR or scBL) and *sc4* is a scanner on the surface from which material is being copied. The precise area to be copied is visualized as follows: the entire surface being copied from is reoriented according to *orient* (see Key in Table B.1) and then supeimposed on the surface to be changed in such a position that *sc3* and *sc4* coincide. It is that area which now falls on the rectangle defined by *scTR* and *scBL* which is used: this area is first transliterated according to table n if n is given, and then it is ANDed, ϕRed or written into the rectangle, accordingly as *mode* is AND or ϕR or WRITE. Peculiar and unexpected patterns may result if the two rectangular areas involved are overlapping areas on the same surface, unless they are exactly the same area and *orient* is ST. The above sample instruction involves a 90-degree rotation to the left (counterclockwise). Thus, if scanner D is located as shown:



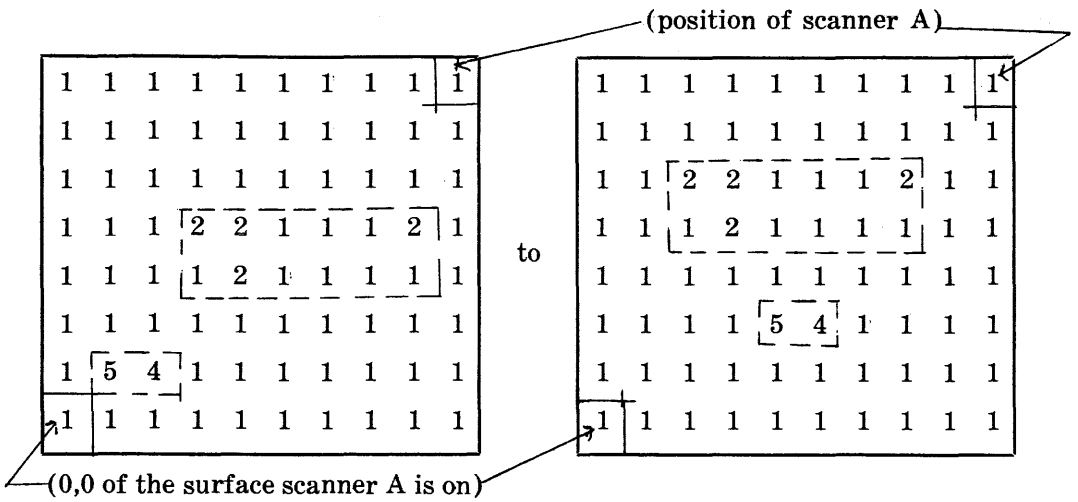
the effect would be to change the rectangular area



Lines of typed text, such as lines of a movie title, may be centered in a rectangular area by the instruction

CENTER scTR,scBL
(e.g. CENTER A,0)

For this instruction to be effective, all squares bordering on the edge of this area must contain the same numbe, called the "backgound" number. The complete background consists of all rows containing only the background number and, for each horizontal stripe not thus included, it also contains the widest possible rectangle on the right and on the left which contain only background numbers. Remaining subareas thus delimited are called "lines of text." The CENTER operation identifies all lines of text, centers each such line horizontally, and moves all lines together by a shift-up or shift-down to make top and bottom background stripes equal. Thus it would change the area



Two different local operations can be performed throughout a rectangular area. Both consume a relatively large amount of computer time (ca. 10 seconds for the surface AA); therefore the area to which they are applied should be judiciously limited. The first local operation,

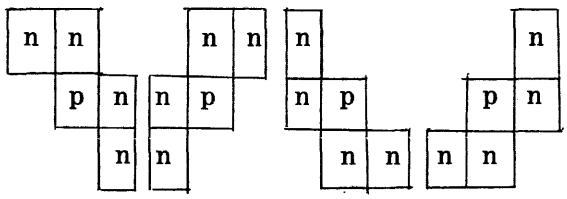
GRØW scTR,scBL,nsl,ns2,ns3,goto
(go to optional)
(e.g. GRØW XX,0,2,3,5,AGAIN3)

changes every number *ns1* which is next to an *ns2* into an *ns3*. The number *ns2* must also be within the area and it must in each instance be immediately above, below, right of or left of the number to be changed. The numbers *ns1*, *ns2* and *ns3* must be three different numbers, and each may be either given explicitly or specified as the name of the scanner sitting on the number to be used. If the instruction contains a *goto*, then on completion of the operation control passes to the *goto* if at least one such number was changed. Two GRØW instructions are commonly used in a loop which causes the region defined by two numbers gradually to "grow" into the region defined by a third by alternately adding incremental layers first of one of its numbers and then of the other. If the loop contains a CAMERA call, the growth process is seen in the movie.

The other local operation, which has the effect of removing sharp corners, is

SMØØTH scTR,scBL
(e.g. SMØØTH A,B)

The operation proceeds by identifying every instance of the four patterns



where all four n's in a pattern must be the same number. For each instance thus found, the p is changed to an n.

4. *Instructions for Dynamic Operations on Rectangular Areas*

Dynamic operations on rectangular areas are those for which a series of pictures are output, according to current camera settings, while the operation is proceeding. As in the case of instantaneous operations on rectangular areas, the affected area must be designated by two scanners (or by one scanner and '0', meaning 0,0 of that scanner's surface).

A rectangular area may have another area of similar size and shape "dissolved" onto it while a series of 36 pictures is being produced. The instruction is

DISØLV scTR,scBL,sc3,pat
(e.g. DISØLV XX,0,ZZ,F)

where scanner *sc3* indicates the top right corner of the area of similar size and shape which is to be dissolved onto the area specified by scTR and scBL. The dissolve is accomplished by dividing the area, which must be at least 6x6

squares in sizes, into 6x6 subareas—possibly with oddly shaped rectangles at the left and bottom. Between any two successive pictures of the dissolve, an additional number of the old picture is replaced by the corresponding number from the new picture in each 6x6 subarea. The order in which new numbers are thus introduced is specified by the parameter *pat* which must be one of the following:

F fade: quasi-random order
SI spiral in (simultaneously in all 6x6 subareas)
SØ spiral out (simultaneously in all 6x6 subareas)
R wipe to right (simultaneously in all six-column-wide sections)
L wipe to left (simultaneously in all six-column-wide sections)

"A "zoom" effect may be approximated by gradually magnifying horizontally and vertically, by an integral factor, the pattern of numbers within a rectangular area. This is accomplished by the instruction

ZØØMIN scRT,scBL,fctr
(e.g. ZØØMIN YY,0,3)

where *fctr* is the magnification factor, an integer from 2 to 6. During the ZØØMIN operation, the approximate center of the pattern is fixed and material is lost off all four edges.

An approximate inverse of ZØØMIN is accomplished by the instruction

REDUCE scTR,scBL,fctr
(e.g. REDUCE A,B,5)

where *fctr* is the factor of reduction, again an integer from 2 to 6. Reduction is accomplished by repeated deletions of rows and columns and by repacking toward the center those remaining. Vacated area around the periphery is filled in by repetitions of rows and columns originally just within the periphery of the area.

A unidirectional magnification is accomplished by the instruction

STRECH scTR,scBL,dir,fctr
(e.g. STRECH A,B,UP,2)

which holds one edge of the pattern fixed and "stretches" the pattern—by duplicating rows or columns—in the indicated direction until (1/fact)th of the original pattern covers the entire area, with each of these rows (columns) repeated *fctr* times. The direction *dir* must be UP, RIGHT, DØWN, or LEFT, and the factor *fctr* must be an integer from 2 to 6.

Finally, an approximate inverse of STRECH is provided by the instruction

PRESS scTR,scBL,dir,fact
(e.g. PRESS ZZ,0,DØWN,3)

which causes the contents of the area to be compressed against one side of the area—by deletion of rows or columns and repacking in the direction *dir*. The vacated area is filled with repetitions of the row or column just within the edge from which motion occurs. As before, *dir* must be UP, RIGHT, DØWN, or LEFT, and *fctr* is the factor of compression, and integer from 2 to 6.

APPENDIX C

An Example of Movie Language Coding

The actual program which produced Fig. 2a is here given as an example of movie language coding. It consists of five parts: definitions of new macro-instructions, coding for composing the picture, coding for outputting the picture, closed subroutines, and descriptions of text to be typed. Particular attention should be paid to the first section, which illustrates how the programmer devices his own macro-instructions on a still higher level, instructions which are either more powerful or which are more specifically suited to a particular task.

A new macro-instruction of wide application, BENTLN, is used for drawing any line which consists of a series of straight line segments, such as the two braces of Fig. 2a. BENTLN is defined in terms of SEGMNT, which compiles into a call for the subroutines SEG . . . that actually draws one segment of the line. The new macro-instruction, SURF, on the other hand, was designed specifically for Figs. 2a and 2b. It permits convenient description of the position and size of a surface in the drawing and of labels indicating its name and its size in squares.

The program is here listed, with explanatory comments on the right. (For more complete description of the MACRØ FAP compiler see the Bell Telephone Laboratories, 7090 Programmer's Manual, July 15, 1963):

```
BENTLN  MACRØ   XSTART,YSTART,PARTS (STARTING POINT + LIST OF SEG-
                MENTS)
        THEN    (C,T,BB) (C,X,XSTART) (C,Y,YSTART) (D,T,C)
        IRP     PARTS   ) CØMPILE SEGMNT FOR EACH PART. EACH "PART"
        SEGMNT  PARTS   ) IS A LIST ØF A DIRECTION (U,R,D,ØR L) AND AN
        IRP             ) AMØUNT PLUS AN ØPTIONAL 2ND DIRECTIØN AND
        ENDM            ) AMØUNT. END ØF BENTLN DEF.
*
SEGMNT  MACRØ   DIR1,AMT1,DIR2,AMT2             (DIR2 AND AMT2 ØPTIØNAL)
        IFF     0,/CRS/DIR2                     IF DIR2 PRESENT THEN
        THEN    (D,DIR2,AMT2)                   CØMPILE THIS LINE
        THEN    (D,DIR1,AMT1) (QQ,P,SEG . . .)  CØMPILE CALL TØ SEG . . .
        ENDM                                    END ØF SEGMENT DEFINITIØN
*
SURF    MACRØ   X1,Y1,WIDTH,HEIGHT,NAME,SIZELN,SHIFT (SHIFT ØPTIØNAL)
        PLACE   B,BB,X1,Y1
        THEN    (A,T,B) (A,R,WIDTH) (A,L,1) (A,U,HEIGHT) (A,D,1)
        PAINT   A,B,WRITE,2                     PAINT RECTANGULAR AREA
                                                WITH 2'S
        THEN    (C,T,B,) (C,U,1) (C,R,1) (D,T,B) (D,D,8)
        TYPE    NAME,C,8X11,1,1,WRITE,0         TYPE NAME WITH 0's
        CENTER  A,B                             CENTER IT
        IFF     0,/CRS/SHIFT
        THEN    (D,R,SHIFT)                     SHIFT SIZE LABEL RIGHT
        TYPE    SIZELN,D,5X7,1,1,WRITE,7        TYPE SIZE LABEL
        ENDM                                    END ØF SURF DEFINITION
*                                               END ØF DEFINITIONS OF NEW
                                                MACRØS
        FILM                                    BEGINNING ØF PRØGRAM
        FRAMES  ϕ                               (NØ ØUTPUT DESIRED DURING
                                                TYPING)
        PAINT   BB,ϕ,WRITE,ϕ                    CLEAR SURFACE BB TØ 0'S
        BENTLN  46,173( (L,3,D,2) (D,18) (L,3,D,2) (R,3,D,2) (D,17) (R,3,D,2) )
        BENTLN  95,173( (R,3,D,2) (D,46) (R,3,D,2) (L,3,D,2) (D,40) (L,3,D,2) )
        PLACE   C,BB,8,167                      (TWØ BRACES NOW DØNE)
        TYPE    ØRSUR,C,5X7,1,2,WRITE,7         TYPE "ØR SURFACE . . ." NØTE
        PLACE   C,BB,10,60
        TYPE    FIG2A,C,5X7,1,3,WRITE,7         TYPE "FIG. 2A . . ." CAPTIØN
        PLACE   C,BB,103,125
TPØR    TYPE    TEXTOR,C,5X7,1,2,WRITE,7        TYPE "ØR"
        IF      (C,L,120)T(C,X,152)TPØR         MOVE RIGHT, TYPE "ØR" AGAIN
        SURF    52,140,40,28,TEXTAA,252X18
        SURF    52,90,40,28,TEXTBB,252X18       DØ THE 4 SURFACES
        SURF    110,100,40,56,TEXTVV,252X36
        SURF    160,110,80,28,TEXTCC,504X18,27
        BØRDER  BB,0,2,WRITE,7
*                                       BEGIN CAMERA SETTINGS FØR ØUTPUT
        AIM     BB                      AIM CAMERA AT SURFACE BB
        FINE                            ØUTPUT ENTIRE SURFACE BB (252X184)
        FILTER  5                       ) TRANSLITERATE 0'S TO BLANKS (60)8
        TABLE   5,60,1,72,3,4,5,60,54   ) AND 7'S T0 ASTERISKS (54Ø8)
        FRAMES  150                     DA 1 FRAME, WITH "150" ON FRAME LINE
```

```
         SAMPLE   1                  ) PRØDUCE ØN PRINTER ALL ØDD
         LINES    17777777777       ) CØLUMNS OF ALL ØDD LINES
         CAMERA                       PRØDUCE PICTURE
         TRA      FINISH
*                                     BEGINNING OF CLØSED SUBRØUTINES
SEG . . . LINE    C,D,WRITE,7,1,5000 ) SUBRØUTINE FØR DRAWING A
         THEN     (C,T,D)QQ          ) STRAIGHT SEGMENT ØF A LINE
ØRSUR    BCI      2,*( ØR            ( DESCRIPTIØNS OF TEXTS      )
         BCI      2,*/SUR-           ( TO BE TYPED BEGIN HERE     )
         BCI      2,*/FACES          ( AND CØNTINUED TØ THE END.  )
         BCI      2,*/ ØF
         BCI      2,*/FIG.
         BCI      2,*/ 2*)B.*.
TEXTAA   BCI      1,AA*.
TEXTBB   BCI      1,BB*.
TEXTCC   BCI      1,CC*.
TEXTVV   BCI      1,VV*.
252×18   BCI      2,252*)×184*.
252×36   BCI      2,252*)×368*.
504×18   BCI      2,504*)×184*.
TEXTOR   BCI      1,Ø*/R*.
FIG2A    BCI      8,*(FIG. *)2A. *NAMES AND SIZES OF SURFACES
         BCI      8,*/LARGE ENØUGH TØ HØLD CØMPLETE FINE-
         BCI      8,*/RESØLUTIØN MØVIE FRAMES (252 × 184).
         BCI      8,*/*ALTERNATE USES ØF TØTAL 7090 STØRAGE
         BCI      8,*/AREA ARE INDICATED, INCLUDING THE USES
         BCI      8,*/ØF REGIØN *A*A IN WAYS SHØWN IN *FIG. 2B.*
```

SIMULATION OF BIOLOGICAL CELLS BY SYSTEMS COMPOSED OF STRING-PROCESSING FINITE AUTOMATA*

Walter R. Stahl,† Robert W. Coffin,‡ and Harry E. Goheen§

INTRODUCTION

In the last few years enormous progress has been made in clarifying the operational mechanisms of living cells. It has been established beyond reasonable doubt that all aspects of cell activity are controlled by sequences of elementary genetic units. A comma-free triplet coding in the four-letter alphabet of DNA is transcribed on RNA and causes the formation of sequences of amino acids, which make up polypeptides and proteins. Various theories of transcription control for such systems are now under study. Recently, synthetic nucleic acid (RNA) chains have been fed into the cell machinery, thus demonstrating that protein synthesis can be controlled artificially. Numerous finer details of the problem could be mentioned (see Crick,[9] Nirenberg,[31] Rich,[39] Waddington,[57] and Anfinsen[1]) but shall not be considered in this report.

There arises the question of what type of mathematical modelling method is best suited for simulation of molecular genetics. In the past numerical models, based on chemical kinetics expressed in terms of simultaneous differential equations, have usually been applied. Impressive results were obtained by Chance et al.,[7] Garfinkel,[13] Hommes and Zilliken[18] and others. However, it has also become clear that simultaneous solution of hundreds or thousands of differential equations, many of whose coefficients probably cannot be measured experimentally (see Pardee[32]), poses a difficult problem. Actual cells may contain thousands of different genes and hundreds of thousands of synthetic units. Major questions of solvability and stability of such systems must be faced in an attempt to model a complete cell.

The present report describes a fundamentally different approach to the problem of simulating a cell, some aspects of which were reported earlier in Stahl and Goheen.[45] Since genes and proteins are representable as linear chains or strings, it is proposed that molecular mechanisms of cells be simulated by string-processing finite automata. In this model strings representing DNA, RNA, proteins and general biochemicals are subjected to controlled copying, synthesis into longer strings and breakdown into shorter ones, with use of what may be called "algorithmic enzymes." A major property of these logical operators is that they are combinable into large systems with complex properties.

The materials below deal in turn with a new computer simulation system for studying finite automata, the properties of algorithmic enzymes, experimental studies with systems of the latter and lastly with some questions of solv-

* This report was prepared under the support of Grant GM-11178 of the National Institutes of Health.

† Scientist, Oregon Regional Primate Research Center, Beaverton, Oregon, and Associate Professor, Department of Mathematics, Oregon State University, Corvallis, Oregon.

‡ Chief Programmer, Department of Biomathematics, Oregon Regional Primate Research Center, Beaverton, Oregon.

§ Professor, Department of Mathematics, Oregon State University, Corvallis, Oregon.

ability for model cells defined entirely by automation-like enzyme operators.

THE AUTOMATON SIMULATION SYSTEM

The quintuplet command code proposed by Turing is used in the programming system and well described in Turing,[50] Davis[10] and elsewhere. Turing's device was designed for proofs of computability and in principle requires an infinite tape and infinite number of recursive steps for such demonstrations. This circumstance does not prevent one from using the quintuplet code for general purpose simulation of automata. The late John von Neumann[55,56] pointed out that the Turing Machine represents a means of programming or simulating *any* algorithm, as well as for computability demonstrations. McNaughton[26] has emphasized that the Turing Machine should be considered as the most general of automata.

A compiler based on the Turing quintuplet notation (but not really modelling the Turing Machine) has been designed and is described in Coffin, Goheen and Stahl.[8] Simulation of a considerable number of different automata on the system, including ones for pattern recognition, has revealed that a computer program for modelling of automata is a useful research tool, which may find practical applications when it is desired to use a "variable programmed automaton." Results substantiating this conclusion are reported in Stahl, Coffin and Goheen[44] and Stahl, Waters and Coffin.[46] A special compiler, constituting an "automaton simulation program," was written for a SDS-920 computer and has processing rates of up to 10,000 quintuplet commands per second. Rates of over one million automation commands a second should be possible with presently known technology. A number of special provisions have been included for automatic sequencing of different circumscribed algorithms or automata presented as lists of quintuplets, debugging, selective printout of string during simulation runs, and so forth.

A Turing compiler should not be evaluated on the basis of the inefficient operation of most Turing Machines described in the literature to date. The authors are using individual Turing Programs (algorithms) exceeding 1700 quintuplets in size and a complete system (namely, the algorithmic cell), which includes over 43,000 quintuplets. Interesting results have been obtained for the problem of recognition of hand-printed letters (A-Z) and shall be reported elsewhere (Stahl, Waters and Coffin[46]). Naturally, automaton simulation has a special range of application, as do research compilers such as LISP, IPL-V, or COMIT.

The Turing Machine is a device which operates on individual symbols presented on a single long tape, along which a reading head moves left or right, one cell square at a time. A capability is provided for erasing and writing individual symbols and for recording the "state" of the Turing Machine, which defines uniquely its response to a particular viewed symbol.

Only one type of program command is used and consists of a quintuplet (or matrix table with output triplets of symbols), which usually appears as follows: symbol scanned, state of machine, new state, motion (right—R, left—L and stay in place—P) and symbol to replace existing symbol before motion is carried out. A quintuplet such as 12 A:15RB is read "if in state 12 and A is viewed, then replace A by B, go to state 15 and move right one square." A final logical halt of the automaton takes place on such an entry as 26* :26P*, which is read "if asterisk is seen in state 26, remain in place, stay in state 26 and do not alter asterisk."

In principle, the Turing Machine must have available an infinite tape and amount of time, but precisely the same notation can be used with finite automata and this has been done by such authors as McNaughton,[26] Myhill,[30] Trakhtenbrot[49] and others. The quintuplet command structure need not in itself connote the extended and often inefficient "shuffling" operation of the classical Turing Machine.

ALGORITHMIC ENZYMES

The concept of a Turing quintuplet code may be illustrated with a simple but biologically provocative example in which a finite automaton simulates a lytic enzyme that breaks down strings. Table I is a quintuplet program for an "automaton enzyme" which lyses strings in the alphabet (AGCT), representing the four nucleic acid bases adenine, guanine, cytosine and thymine. A typical input tape into the

automaton using this code might be

$$\ldots \phi\ \phi = A\text{–}C\text{–}G\text{–}C\text{–}C\text{–}T\text{–}T\text{–}A\text{–}G\text{–}C\text{–}A = \phi\ \phi \ldots \quad (1)$$

in which "ϕ"—empty cells, "="—start of string, "–"—bond between letters.

Table I

TURING PROGRAM FOR A SIMPLE LYTIC ENZYME

1 ϕ:1Rϕ	2 T:2RT
1 =:2R=	2 /:2R/
2 =:2P=	3 =:3P=
2 A:2RA	3 –:2R/
2 C:2RC	3 /:2R/
2 G:3RG	
2 –:2R–	

Following a single left to right pass the string in (1) will be converted into

$$=A\text{–}C\text{–}G/C\text{–}C\text{–}T\text{–}T\text{–}A\text{–}G/C\text{–}A= \quad (2)$$

in which every bond immediately to the right of a G, regardless of what symbol is next to it on the left or right, is converted into an "open bond" (/).

Operation commences at the left end of the string. The empty symbols (ϕ) are passed over by entry 1 ϕ:1Rϕ. When the left end-marker (=) is seen control passes to state 2. In state 2 all symbols except G (namely, A,C,T and –) are simply skipped over, as in 2 –:2R–. If G is seen, by entry 2 G:3RG, control passes to state 3, which next encounters a "bond" and converts it to an "open bond" using entry 3 –:2R/. Provisions are also made for skipping over any existing open bonds, as in entry 2 /:2R/. Stopping occurs in state 2 or 3, on an entry such as 3 =:3P=.

Table II is a sample of coding for a string synthesizing finite automaton, which was described in Stahl and Goheen[45] and is the prototype for the algorithmic enzymes noted in this paper. The cited work also includes automata for copying and complementary copying of strings (as in DNA transcription), and for more complex types of lysis.

In general, the construction of quintuplet programs for automata is straightforward. It is noteworthy that they are truly interchangeable because of the very standard format. It is clear, however, that string-processing enzymes might also be represented by other types of automata, such as Wang's[58] modified Turing Machine and that it would be entirely feasible to design special compilers that accept a "state-symbol" table.

While the lytic enzyme of Table I was given principally as an example, it is interesting to note that reconstruction of a parent protein string following the action of several lytic enzymes is an important problem today for nucleic acid and protein analysis. Rice and Bock[38] have pointed out that application of three specific lytic enzymes, which split "next to" only three of the four specific bases in DNA, does not allow a unique reconstruction of the parent chain. This is an excellent example of algorithmic unsolvability arising in a biological context, and moreover even in very classical form, namely, solution of a "word problem" by algorithmic methods.

It must be emphasized that the lytic enzyme of Table I in no way models the physiochemical properties of any real enzyme that might perform the indicated lysis, and only simulates the string-processing aspects of the enzyme action. This type of model is somewhat comparable to the McCulloch-Pitts[25] imitation of neurons by the threshold Boolean "logical neuron," in that both model systems are rather gross from the biological viewpoint, but involve a consistent mathematical methodology. The McCulloch-Pitts neuron can be combined into large systems, such as perceptrons, and much the same step has been taken with algorithmic enzymes. The main problem in biological modelling is probably that of finding well-defined mathematical methods which can be applied with profit to the biological system.

Table II
TURING PROGRAM FOR CONDITIONAL STRING SYNTHESIS

1=:2R=	6=:7L=	10c:12Rc	15=:16L=	19c:19Lc
1ϕ:1Rϕ	6*:7L8		15*:16L*	19e:19Le
1a:11Lx	6a:8Ra	11=:13R=	15a:17Ra	19x:19Lx
	6b:8Rb	11*:11L*	15b:17Rb	
2=:2P=	6c:8Rc	11ϕ:11Lϕ	15c:17Rc	20*:20R*
2*:2R*		11a:11La		20ϕ:20Rϕ
2ϕ:2Rϕ	7=:9R=	11b:11Lb		20a:20Ra
2a:2Ra	7*:7L*	11c:11Lc	16=:20R=	20b:20Rb
2b:2Rb	7ϕ:7Lϕ	11e:11Le	16ϕ:19L*	20c:20Rc
2c:2Rc	7a:7La		16a:16Lϕ	20e:21rϕ
2e:3Le	7b:7Lb	12=:12P=	16c:16Lϕ	20x:20Rϕ
	7c:7Lc	12*:9R*		
3=:4R=	7e:7Le	1qϕ:12Rϕ	17=:18L=	21=:22R*
3*:3L*		12a:12Ra	17*:13R*	21*:21R*
3ϕ:3Lϕ	8=:8P=	12b:12Rb	17ϕ:17Rϕ	21ϕ:21Rϕ
3a:3La	8*:4R*	12c:12Rc	17a:17Ra	21a:21Ra
3b:3Lb	8ϕ:8Rϕ		17b:17Rb	21b:21Rb
3c:3Lc	8a:8Ra	13ϕ:17Rϕ	17c:17Rc	21c:21Rc
	8b:8Rb	13a:14Ra		21e:21Re
4ϕ:8Rϕ	8c:8Rc	13b:17Rb	18=:18P=	21x:21Rϕ
4a:8Ra		13c:17Rc	18*:18L*	
4b:5Rb	9ϕ:12Rϕ	13e:17Re	18ϕ:18Lϕ	22ϕ:23Ra
4c:8Rc	9a:10Ra	13x:17Rx	18a:18La	23ϕ:24Rc
4e:8Re	9b:12Rb		18b:18Lb	24ϕ:25Ra
	9c:12Rc	14=:18L=	18c:18Lc	25ϕ:26L=
5*:4R*	9e:12Re	14*:13R*	18e:18Le	
5a:8Ra		14a:17Ra	18x:18La	26=:2R=
5b:6Rb	10=:1L=	14b:17Rb		26*:26Lϕ
5c:8Rc	10*:1L*	14c:15Rc	19=:20R=	26ϕ:26Lϕ
	10a:12Ra		19*:19L*	26a:26La
	10b:12Rb		19ϕ:19Lϕ	26b:26Lb
			19a:19La	26c:26Lc
			19b:19Lb	26e:26Le

The concept of studying cells algorithmically was probably implied in von Neumann's[55] work on self-reproduction of structures composed of finite automata. This model has been extensively analyzed and extended by Burks[4] and Moore.[29] The growth and stability of automaton-like arrays is considered in Lofgren,[22] Ulam,[54] Eden,[11] Blum[3] and others. Rashevsky[37] suggested the application of the Markov "Normal Algorithm" (A. A. Markov[24]) to the genetic codes, but did not define any complete cell model. Pattee[35] proposed that a simple automaton could produce long biological chains of a repetitive kind. Induction and repression mechanisms in cells are analyzed by Jacob and Monod.[20] Sugita[47] and Rosen[40] explore the

Boolean logical representation of cells. Turing himself wrote a paper[51] entitled "The Chemical Basis of Morphogenesis," but used differential equations rather than algorithm theory in this study. Soviet workers such as Frank[12] and Pasynskiy[34] discuss some general aspects of cellular control theory. Lyapunov[23] proposes the systematic study of interacting automata subjected to executive hierarchies of control and Medvedev[27] deals with errors in genetic coding.

An "algorithmic cell" is defined in this report as a system of string-processing finite automata, representing enzymes, together with the cell contents, identified as strings. Smaller biochemicals which are not as obviously strings as DNA or proteins may be coded as strings. This is done routinely in ordinary chemical nomenclature. Furthermore, the linearly-coded enzymes (proteins) must be able to recognize any normal biochemical in the cell, and this may be interpreted to mean that some sort of a string coding of biochemicals is possible. Biochemists represent DNA, RNA and proteins in an associative symbolic notation, but this fact has not been subjected to mathematical interpretation. That is, molecular biologists now assume that the four DNA bases, the four RNA bases and the 20 or so pertinent amino acids can and should be presented as symbols in an associative (parentheses free) linear string notation. Much progress has been made recently in showing the details of the nucleic acid to protein coding (Nirenberg[31]). It is noteworthy that a normal algorithm is implied in the DNA triplet to amino acid substitution coding, e.g., CAC →Pro. (cytosine-adenine-cytosine codes the→ amino acid proline). Although the DNA to protein coding is understood, almost nothing is known about the grammer, syntactical relationships or programming techniques used by the cell during its strictly deterministic, and therefore algorithmic, growth, differentiation and functioning.

Figure 1.1 is a flow chart of string syntheses taking place in a typical cell model of the type suggested in this report. At present 43 separate threshold algorithmic enzymes are used and produce a total of nine different final products. The complete model cell consists of the enzyme operators stored on magnetic tape, a 2000 symbol memory which has a place for each string and its numbers at any moment of operation, and three "housekeeping programs" (written entirely in quintuplets representing specialized finite automata) which perform addition, generation of pseudo-random variables, totalling operations, counting, etc. While this type of programming is obviously not as efficient as a conventional computer methods for straightforward arithmetic operations, it is a consistent automaton simulation methodology and very flexible. For example, total cell size may be controlled by simply introducing a special automaton that changes rates of "diet" letter entry as a function of total cell string count. A complete growth experiment with an algorithmic cell requires about 30-45 minutes and involves hundreds of thousands of individual enzyme steps.

The algorithmic enzyme used in the cell model is more complex than the one defined by Table II and functions as follows. Each enzyme is stored as a program of about 1000 quintuplets on magnetic tape and called into the core memory by the compiler program when a preceding enzyme or a "housekeeping automaton" writes in a "call number" in a specific region of the memory tape. As the first step, threshold checks are made of "energy" represented as a simple string and of one or two control strings, by interrogation of the binary numbers associated with these strings. If these thresholds are met it is next ascertained that the substrates (input materials) for the specific enzyme are available in sufficiently high quantities and that the final product is not already present in excessively large amounts. If these added threshold conditions are met, the enzyme functions, producing a fixed quantity of its product, while removing appropriate quantities of the materials that went into the latter. After this is done the algorithmic enzyme writes in another "call number" which controls which housekeeping program or other enzyme shall function next.

In Figure 1 the enzymes are identified by number, as #113, and the "k" value indicates synthetic rate for a unit cycle, while the strings along the flow lines are the control substances.

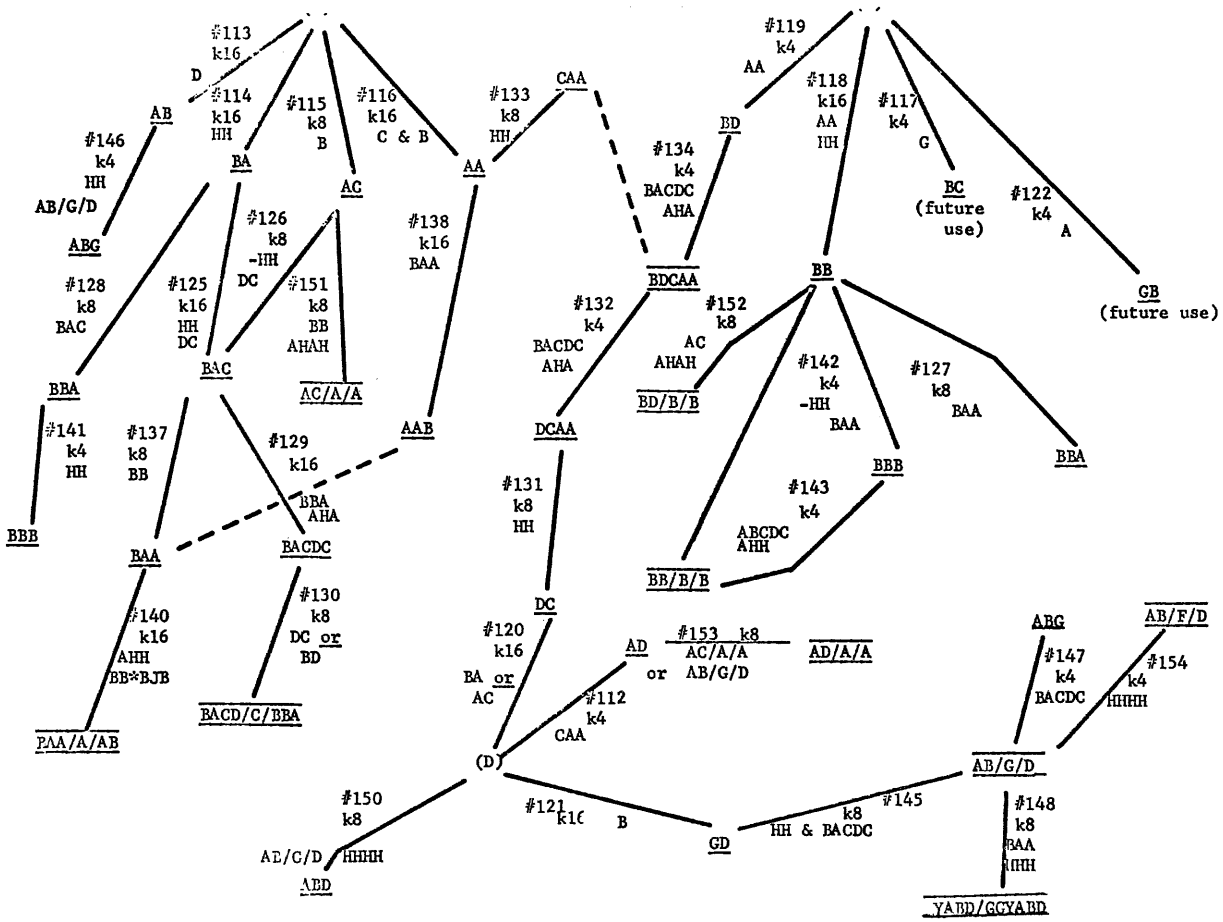


Figure 1.1. Main String Synthesês (each pathway is shown with algorithmic enzyme call number, rate and control substances).

A complete set of specifications for the enzymes shown in Figure 1 will be published elsewhere. It can be seen from Figure 1 that certain "inducer strings," such as AHA, HHH, etc., control the action of sets of coordinated reactions. Moreover, synthesis of these "control strings" is subjected to induction or repression as a function of certain special strings that may be supplied from the outside, as by an adjacent model cell, or manufactured in the cell if levels of "metabolite strings" meet certain arbitrary thresholds. A reasonable correspondence can be drawn between the activities of the algorithmic cell model and known basic cell activities, including blocking and unblocking of DNA, formation of messenger RNA, synthesis of enzymes and proteins on ribosomes, production of metabolites by enzymes, etc.

The flow chart of Figure 1 is entirely arbitrary and does not attempt to represent any actual cell. It must be noted that a real cell may have tens of thousands of enzymes and genes, and that firm quantitative data is available at present only for limited sets of enzymatic reactions. Nevertheless, one may study the basic mathematical problems presented by string-synthesizing automata. The final products of the algorithmic cell model, as shown in Figure 1 were designed to combine into several different two-dimensional arrays. An example of a unit in such an array is given in Figure 2. A "complementation algorithm" is applied to the product strings. This states that any two strings will stick together or polymerize providing that they have at least three complementary bonds, such as A with B, B with A,

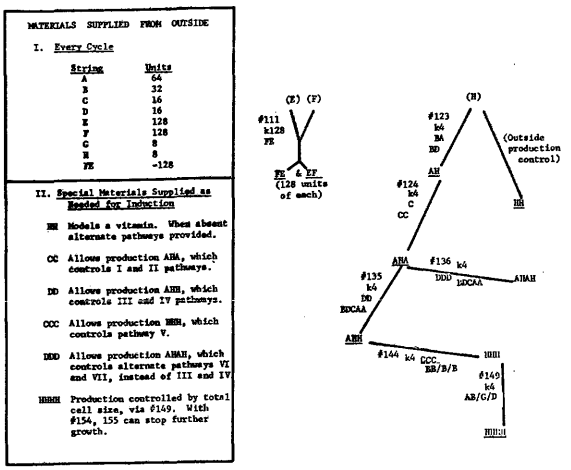
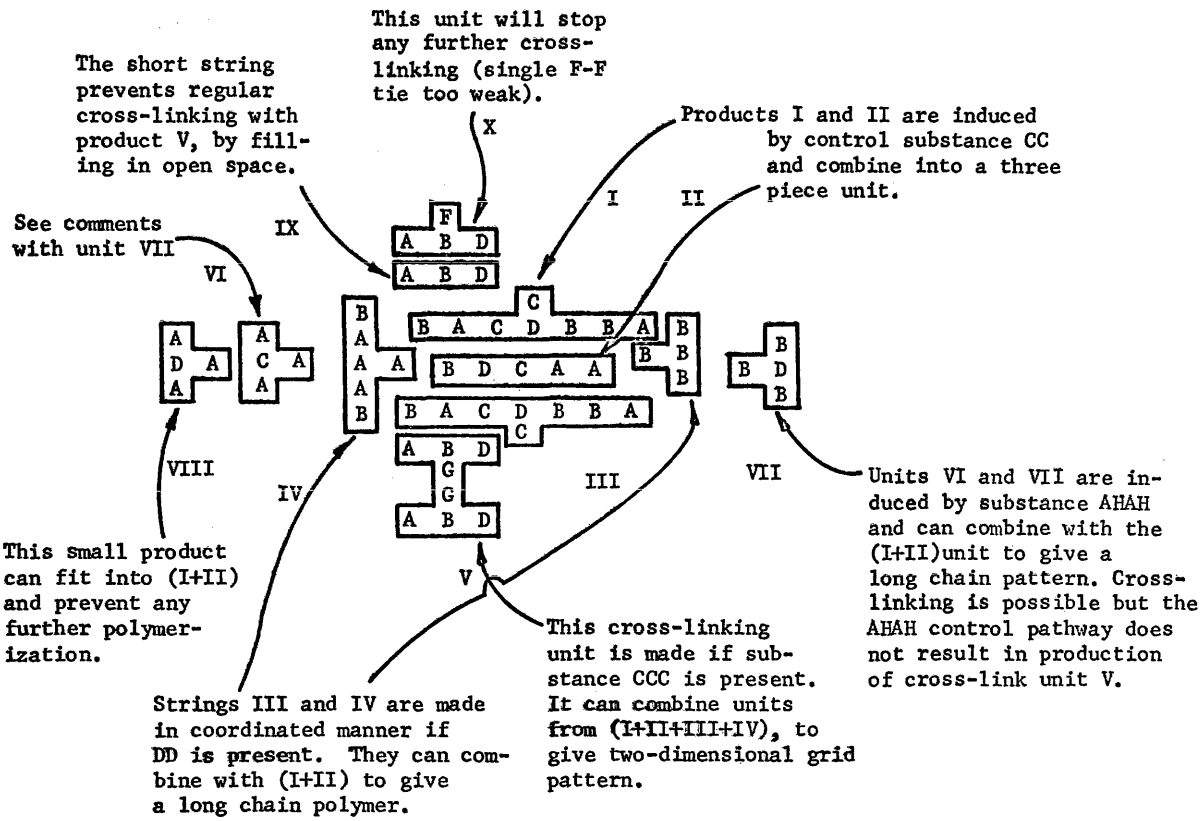


Figure 1.2. "Diet," "Energy," and "Control" Substances.

C with D, etc. Presence of a non-complementary bond negates two complementary bonds. This may be considered to be still another kind of discrete threshold action that generates complex entities from simplier units.

The control strings are so chosen that they regulate the coordinated formation of one or more final products that polymerize together. There are sufficient product strings so that different types of arrays are possible and may lead to long individual chains, cross-linked chains, circumscribed blocks of strings and even grids with a special border that limits further growth. All the operations leading to these results are defined by firm algorithms. However, pseudo-random variables are added to generation rates and this results in less "mechanical" action.



COMPLEMENTATION ALGORITHM: Any two strings which collide are assumed to link or "polymerize" if three or more complementary bonds are present (A-B,B-A,C-D, etc.). A non-complementary bond negates two complementary ones. A single collision is assumed to test all bonds.

Figure 2. Two-Dimensional Configurations Associated with String Products of the Algorithmic Cell Model.

EXPERIMENTAL STUDIES WITH THE ALGORITHMIC CELL

In a typical run the cell model reaches equilibrium after about some 40–50 passes through the sequence of algorithmic enzymes, whose activity or inactivity is determined by induction and repression conditions. During the run printouts of quantities of each string are obtained periodically, as on every second or fourth unit pass. Cycles of operation, totals of strings of given sizes, grand totals and total use of energy are also recorded and used for assessing the efficiency of a given cell control system.

Among the specific experiments that will be performed with the cell model are included 1) curtailing or completely limiting input substances; 2) introducing a "leakage" of one or more substances; 3) limiting energy use or "suffocating" the cell with unremoved products; 4) suddenly eliminating (as if by surgery or trauma) the entire contents of certain reservoirs; 5) adding large random variables to all the generation rates and watching to see if the system will stabilize or oscillate; 6) driving the system with periodic inputs of elementary "diet" letters and noting the transfer functions to other parts of the system; 7) "heating" or "cooling" the cell, by appling a simple constant multiplication factor to rates (k values) of *all* enzymes and then checking stability. Other tests are also possible.

Another experimental study is directed towards finding out approximately what fraction of random changes in the quintuplet commands shall lead to a final cell system that "grows out of control," like cancer. With about 45,000 quintuplets, each capable of many random "mutations," exhaustive testing is not practical even with this small system. However, it is possible to make random changes in representative quintuplets of the more important enzymes and determine whether they cause the algorithmic cell to die, shrink, grow wildly, interact abnormally with adjacent cells (that is, fail to differentiate properly), stop producing one or more of the required final products, etc. All these examples have clear-cut biological counterparts.

Experiments are also planned on competition between two or more algorithmic cells, which are subjected to random mutations in selected enzymes. This is done by having the cells draw upon a common diet and share a pool of some, but not all, strings. It is then possible to model experimentally a kind of simulated natural selection, in which the cell that first reaches final "adult" size is selected over lagging ones.

Self-reproduction, as such, can be modelled with the above system, and would involve having the algorithmic enzyme quintuplets listed on a long "gene" string, which is copied at time of reproduction. Self-reproduction becomes possible when the length of the "gene" coding is sufficient to define a certain minimum number of operational enzymes which account for cell structure, basic metabolism, gene copying and over-all control of the entire simulated cell system. This type of self-reproduction differs in certain fundamental respects, i.e. exclusive use of strings from the "kinematic" or "grid" reproduction considered by von Neumann,[55] Burks[4] and Moore,[29] and also does not resemble very closely the geometric growth systems of Ulam[54] or Eden.[11] The details of an existence demonstration for self-reproduction in an algorithmic cell model shall be considered elsewhere.

As was noted above the final string products of the cell model combine or polymerize to form rather intricate two-dimensional arrays, somewhat reminiscent of known protein polymers such as collagen or muscle fibers. Following polymerization, which at present is modelled by hand but could be represented on a digitally-controlled projection screen, certain specific sequences of letters may be read off around the margins of the two-dimensional arrays which may be chains, cross-linked chains, grids or irregular crystal-like structures resembling those described in Ulam.[54] Future studies will make use of algorithmic pattern recognition strategies to determine when an array has reached its final desired configuration and also what may be done to repair such an array if it has been injured, i.e., a corner chopped off a grid. In the biological world there is a clear counterpart to these abstract studies in the ability of cells and tissues to regenerate missing parts using strictly determinate procedures which are clearly coded in their gene sequences.

The described system is a sequential, not parallel, simulator. In the real cell it is known that tens of thousands, if not millions, of copying and synthetic activities may proceed in parallel. It is possible to mimic the effects of many identical enzymes acting simultaneously by increasing rates of production, and to minimize real-time non-simultaneity problems by using a rather short unit time period, during which no one synthetic chain is produced in large quantity. However, future studies will explore various ways of simulating a parallel string-processing system.

As noted, the above system does not attempt to model any real cell or the properties of real enzymes in a physiochemical sense. Recent studies of enzyme action (Labouesse et al.,[21] Monod et al.[28]) have shown that they are formidable objects from the physiochemical standpoint, because they are composed of hundreds of individual amino acids and have a complex three-dimensional structure. They are now believed to function, at least in part, through physical dislocations of certain enzyme regions after a given substance becomes attached to a receptor location. One must keep in mind that these remarkable logical machines, which can be characterized as "coiled up" or "multiply connected" strings, are determined solely by the linear array of their amino acids, which are in turn coded by the gene base sequences. Physiochemical models of this "conformation" system have not been very satisfying.

Possibly the central question is choice of type of automaton to simulate an enzyme: it might use numbers, the Boolean variables 1 and 0, or letter strings, as discussed above. The differential equations models can be subsumed under analog-type systems that process real numbers (concentrations and rates). Sugita[47] and Rosen[40] suggest a Boolean logical model of cellular biochemical mechanisms. Arbib[2] and others have demonstrated the algorithmic equivalence of the Turing Machine, finite automaton and McCulloch-Pitts net, so the Boolean method and general string approach are theoretically equivalent. However, it appears that string-processing methods are more appropriate for contemporary molecular biology, which already uses biochemical strings (nucleic acids and amino acids in proteins) as a basic tool.

ALGORITHMICALLY UNSOLVABLE PROBLEMS FOR AN ALGORITHMIC CELL

One of the most interesting conclusions which can be drawn from the above model is that a cell composed of string-processing enzymes, coded by strings of genetic symbols, has definite limitations on what it can do, i.e., there are many plausible situations for which it would have no adaptive algorithm, or "definite procedure," to use Turing's own term.[52]

One has to consider what kind of logical problems may be solved by a well-defined, circumscribed system of finite automata of the above type, i.e., ones which may copy strings, synthesize or lyse strings while obeying a principle of "conservation of letters" (with the exception of some random variables imposed on reaction rates), and also perform certain types of blocking and unblocking actions, simulating cellular induction and repression. In addition the enzymes may be combined into structures that have barrier properties, i.e., which allow a string to pass from one region to another. Enzymes may also be linked into fairly complex subassemblies. Insofar as is known today, the cell does not contain specialized molecular automata that perform binary logic, arithmetic, compute or "think" in any ordinary sense.

A system of 43 algorithmic enzymes has logical abilities that are qualitatively far more complex than those of an individual enzyme, but still circumscribed. Consider, for example, what happens when an algorithmic enzyme contained in a virus attacks an algorithmic cell. This virus enzyme may lyse an important string of the cell, or use normal biochemical strings in the cell for its own purposes. The original cell may or may not be able to resynthesize the lysed products or to produce enough of them to overcome its losses. It may or not contain any precoded lytic enzymes that can dissolve the enzymes or other constituents of the virus.

What is much more basic, perhaps, is that there may exist no algorithm by which the cell system can identify a virus on the basis of its genetic or protein sequences, or from its initial

effects on itself, which may result in new products that the cell is unable to recognize. If a cell attempted to "compute" the string composition of a new enzyme (there is no evidence that cells do this) designed specifically to attack a virus, it would also have to solve the following problems, which have well-defined algorithmically unsolvable counterparts in Turing Machine theory (see Davis,[10] Trakhtenbrot[49]): 1) will a new proteolytic enzyme (all enzymes are known to be proteins) lyse other desirable proteins of the parent cell (the "applicability problem" of a Turing Machine); 2) will a new proteolytic enzyme lyse itself (the "self-applicability problem" of a Turing code applied to itself —see Trakhtenbrot[49]); 3) will a new enzyme result in a stopping or equilibrium condition of the whole cell (the "stopping problem" of a Turing Machine—see Davis[10]); 4) will a new enzyme give a final cell configuration which is compatible with all stages of coordinated growth and development ("translatability problem," or predicting final cell configuration when given an earlier one); 5) is the "computed" enzyme the most efficient one (this relates to the algorithmically unsolvable problem of minimality of a Turing Machine coding—see McNaughton[26]); 6) will the system with the new enzyme be stable and resist instability during certain possible cellular situations.

Quite similar problems arise if a cell attempts to distinguish a cancer cell from a normal cell, so as to be able to kill it. It is pertinent to ask if the normal defensive cells perform a kind of Turing's Test[53] on potentially cancerous ones, acting in some abnormal way.

As a specific example, suppose a hypothetical cellular automaton were engaged in design of an enzyme of about 150 amino acids (given in associative notation, as Asp-Lys-Glu-Lys . . .) which could attack a virus. One would have to know how this enzyme would function during all past and future stages of growth and differentiation. Information about the latter is present in the genes, but it is not clear how the cell could get access to the desired information without a "dump" from gene memory, which might result in loss of recursive control. In effect, the cell would have to model its own condition during all possible earlier and later states of differentiation. Moreover, the cell is unable to predict what sort of other biochemicals (or viruses) might be present in the environment at later times or the effects of the new potential enzyme on its status in natural selection against other cells. Furthermore, the postulated cellular automaton would have to anticipate the final three-dimensional structure of an enzyme, with its "conformation" (physical folding) properties *in abstracto*, since once a new enzyme was actually made it might lyse the automaton in question or otherwise interfere with normal cell controls.

Reasoning of the above type would suggest that acquired characteristics are not inherited because in the general case a cell attempting to carry out the indicated adaptation is faced with algorithmically unsolvable problems. If direct genetic adaptation were possible it would confer a tremendous advantage and basic reasons why it does not occur should be sought. However, the problem is complicated by the fact that real cells are much more complex than the abstract model of this report and is discussed further in another report (Stahl[43]).

It must be noted that the described model does not adequately simulate either the parallel string processing or real-time processing problems in a cell, in which a given algorithm must be completed in approximate time synchronism with existing developmental algorithms. McNaughton,[26] Rabin and Scott,[36] Burks,[5] and others have considered some aspects of computability for "growing automata" and real-time computability problems. The deliberate design of new algorithmic enzymes might also involve certain questions of solvability for multiple-tape Turing Machines, the isomorphisms (Sorkin[42]) and composition of automata (Glushkov[16]), one-way or back-and-forth reading machines (Schutzenberger[41]), state identification experiments on existing molecular automata (Gill,[14] Ginsberg[15]), and adaptive abilities of automata (Tsetlin[48]).

Algorithmic unsolvability may be referred to Turing Machine theory or to general recursive computation theory. If the latter methodology is used one would presumably reach the conclusion that no decision procedure existed for the choice of amino acids (or gene bases) for a given adaptive problem. In the case of a malig-

nant cell one might say that there was a faulty "transfer of control," or lack of a "stop condition." This type of statement is considered wholly acceptable to cancer specialists at present and is implied in contemporary discussions of the cancer problem. Turing Machine unsolvability has the important advantage of defining several qualitatively distinct unsolvability situations, such as the "stopping problem," "translatability problem," "self-applicability problem," "minimality problem," etc.

The algorithmic cell differs from "learning models" (see, for example, Pask[33]) and certain of the systems described under the term "self-organizing systems" (Yovits and Cameron[59]) because, as contrasted with the latter and the brain, it cannot "learn" its environment, and only responds to it selectively with available precoded algorithms. Interesting problems arise in the study of neurons represented as string-processing systems. Adaptation by the brain does not, of course, imply adaptation by the genes in neurons. Algorithmically unsolvable problems would clearly arise if "new" brain circuits were to be designed by a hypothetical neuronal molecular automaton. For example, Hennie[17] has stated a number of unsolvable problems connected with iterative nets, which may be composed of non-specific threshold neurons.

To state an example, it appears plausible that the deliberated design of a new integrative circuit by a frog brain, for the purpose of detection of some new specific type of a bug projected on its retina and based on an iterative type of circuit, might involve algorithmically unsolvable problems. Pursuing a related line of reasoning, Cannonito[6] has discussed Godel's incompleteness theorem as applied to intelligent machines.

DISCUSSION AND CONCLUSIONS

Careful study of the existing literature in both molecular biology and finite automata theory, including also recent Soviet reports, suggests that the model proposed in this report has not been used before. At present it is in a developmental stage, and is running as described on an SDS–920 computer.

It has been noted above that the model does not attempt to use numerical data from biochemical studies. However, it will be tested in this capacity, with use of finite approximations to enzyme kinetics. Quantitative modelling will become really meaningful when a sufficiently complete and accurate set of parameters is available for an actual cell. Most existing quantitative biochemical models are of the steady-state compartment type and demonstrate changes of equilibria in open systems. The present model is designed, on the other hand, to study coordinated growth and progressive differentiation of a cell. Both types of studies obviously have a place.

Since real cells are so complex, and may include thousands of different enzymes present in many copies, there arises the very real question of what kind of mathematical or algorithmic tool is best suited for the simulation of enzymes and cells. In this report it is suggested that finite automata are a suitable axiomatic tool. While the proposed model is imperfect, it does allow a new and mathematically consistent representation of such extremely characteristic biological phenomena as differentiation, induction and repression controlled by environmental biochemicals, coordinated growth, development of cancers, competition between cells, etc. The need for some new mathematical attack on the problem has been strongly stressed by such senior biologists as Waddington.[57]

As concerns the Turing quintuplet programming technique, it is more efficient than has been supposed and is well suited for study of the properties of systems of automata. It is effective where intensive logical processing of a non-standard variety is to be done on a rather small amount of input information, but, of course, extremely inefficient where much information must be used at one time or for conventional numerical processing. Other reports will take up the useful range of application of the described automaton simulation programming technique.

A model of a general synthetic system may also be of interest in realms other than cellular biology. For example, it may be proposed that in any general production process the addition of a letter to a string can represent a "next processing step," while joining of smaller strings indicates combination of subassemblies

or completed parts. Alternatively, each processing step may stand for some specific informational processing task. Abstract models of this type are interesting for studies of parasitism and suggest, for example, that viruses, tapeworms and human embezzlers might be subsumed under one consistent theory of events in "synthetic productive systems." The analogy is loose and speculative, but provocative.

As another example, it has been known for several years that mammals are able to defend themselves non-specifically against viruses by production of a genetically precoded substance known as "inteferon" (Isaacs[19]). The mode of action of this complex biochemical material is not clear, but when it is discovered it will be interesting to study what algorithm controls its release and how it relates to the unsolvability problems discussed above. There is also much interest in the study of algorithms that may be used to distinguish normal from malignant cells.

Work described in the above is intended as a starting point for studies which deal with specific problems in the design of cellular control algorithms that are stable in the presence of environmental fluctuations, coding errors (mutations) and can to some extent resist parasitic systems such as viruses or cancer cells. Compared with any real cell the model is very crude, but it bears much the same relationship to real tissue cells as the McCulloch-Pitts neuron has to actual brain neurons. The logical neuron has been useful as a firm mathematical tool for study of the over-all properties of brain models and the algorithmic cell can serve in a very similar capacity for systems of cells.

BIBLIOGRAPHY

1. ANFINSEN, C. B. In *The Molecular Basis of Evolution.* John Wiley Science Editions, New York, 1963. p. 115 et seq.
2. ARBIB, M. "Turing Machines, Finite Automata and Neural Nets." *J. Assoc. Comp. Mach. 8*(4) :467–475, 1961.
3. BLUM, H. F. "On the Origin and Evolution of Living Machines." *Amer. Sci. 49*:474–501, 1961.
4. BURKS, A. W. "Toward a Theory of Aumata Based on More Realistic Primitive Elements." *Proc. IFIP Congress, Munich, Germany, 1962.* North Holland Publishing Company, Amsterdam, 1963. pp. 379–385.
5. BURKS, A. W. "The Logic of Fixed and Growing Automata." In *Proc. Int. Symp. Switching Theory. Part I.* Harvard Univ. Press, Cambridge, Mass., 1959. pp. 147–188.
6. CANNONITO, F. B. "The Godel Incompleteness Theorem and Intelligent Machines." In *Proc. AFIPS Spring Joint Comp. Conf., San Francisco, Calif. May 1962.* pp. 71–77.
7. CHANCE, B., et al. "Metabolic Control Mechanisms. Part V. Solution for the Equations Representing Interaction Between Glycolysis and Respiration in Ascites Tumor Cells." *J. Biol. Chem. 235*:2425–2439, 1960.
8. COFFIN, R. W., H. E. GOHEEN, and W. R. STAHL. "Simulation of a Turing Machine on a Digital Computer." *Proc. AFIPS Western Joint Computer Conference, Las Vegas, Nev., November 1963.*
9. CRICK, F. H. C. "On the Genetic Code." *Science 139*(3554) :461–464, 1963.
10. DAVIS, M. D. *Computability and Unsolvability.* McGraw-Hill, New York, 1958.
11. EDEN, M. "A Probabilistic Model of Morphogenesis." In *Symposium on Information Theory in Biology.* H. P. Yockey, R. L. Platsman, H. Quastler, eds. Pergamon Press, New York, 1958. pp. 359–370.
12. FRANK, G. M. "Self-Regulation of Cellular Processes." In *Biological Cybernetics.* N. M. Sisakayan and A. I. Berg, eds. Acad. Sci. USSR, Moscow, 1962. (In Russian). Available as *JPRS 19637.* pp. 40–55.
13. GARFINKEL, D. "Computer Simulation of Steady-State Glutamate Metabolism in Rat Brain." *J. Theoret. Biol.* 3 :412–422, 1962.
14. GILL, A. "State-Identification Experiments in Finite Automata." *Information and Control. 4*:132–154, 1961.
15. GINSBURG, S. "On the Length of the Smallest Uniform Experiment Which Distinguishes the Terminal States of a Machine." *J. Assoc. Comp. Mach. 5*:266–280, 1958.

16. GLUSHKOV, V. M. "Abstract Theory of Automata." *Uspekhi Mat. Nauk, 16*(5): 3–62, 1961. (In Russian).

17. HENNIE, F. C. "Iterative Arrays of Logical Circuits." *MIT Press Research Monographs,* Cambridge, Mass. John Wiley, New York, 1961.

18. HOMMES, F. A., and F. W. ZILLIKEN. "Induction of Cell Differentiation: III. A Quantitative Approach in the Analysis of Induction." *Bull. Math. Biophys. 24:*71–80, 1962.

19. ISAACS, A., H. G. KLEMPERER, and G. HITCHCOCK. "Studies on the Mechanism of Action of Interferon." *Virology 13:*191–199, 1961.

20. JACOB, F., and J. MONOD. "Genetic Regulatory Mechanisms in the Synthesis of Protein." *J. Molec. Biol. 3:*318–356, 1961.

21. LABOUESSE, B., B. H. HAVSTEEN, and G. P. HESS. "Conformational Changes in Enzyme Catalysis." *Proc. Nat. Acad. Sci. 48*(12) :2137–2145, 1962.

22. LOFGREN, L. "Kinematic and Tesselation Models of Self-Repair." In *Biological Prototypes and Synthetic Systems.* E. E. Bernard and M. R. Kare, eds. Plenum Press, New York, 1962.

23. LYAPUNOV, A. A. "Some Questions on the Teaching of Automata." In *Principles of the Design of Self-Learning Systems.* Kiev, USSR. Available as *JPRS 18181,* 1963. pp. 180–185.

24. MARKOV, A. A. "The Theory of Algorithms." *Amer. Math. Soc. Translation Ser. 2. 15:*1–14, 1960.

25. MCCULLOCH, W. S., and W. PITTS. "A Logical Calculus of the Ideas Imminânt in Nervous Activity." *Bull. Math. Biophys. 5:*115–133, 1943.

26. MCNAUGHTON, R. "The Theory of Automata—A Survey". *Adv. in Computers. 2:* 379–421, Academic Press, New York, 1961.

27. MEDVEDEV, Zh. A. "Errors in the Reproduction of Nucleic Acids and Proteins, and Their Biological Significance." *Problems Kibernetikii* No. 9. A. A. Lyapunov, ed. Gos. Izd. Fiz-Mat. Lit., Moscow, 1963, pp. 241–264. (In Russian).

28. MONOD, J., J. P. CHANGEUX, and F. JACOB. "Allosteric Proteins and Cellular Control Systems." *J. Molec. Biol. 6:*306–329, 1963.

29. MOORE, E. F. "Machine Models of Self-Reporduction." In *Mathematical Problems in the Biological Sciences.* Sympos. No. 14 of Amer. Math. Soc., Providence, R.I., 1961. pp. 17–33.

30. MYHILL, J. "Linear Bounded Automata." *WADD Tech. Note 60–165.* Wright Air Development Division, Wright Patterson Air Force Base, Ohio, 1960.

31. Nirenberg, N. W. "The Genetic Code, Part II." *Sci. Amer. 208*(3) :80–94, 1963.

32. PARDEE, A. B. "Biochemistry: Sterile or Virgin for Mathematicians." In *Mathematical Problems in the Biological Sciences.* Sympos. No. 14 of Amer. Math. Soc., Providence, R.I., 1961. pp. 69–82.

33. PASK, G. "The Simulation of Learning and Decision-Making Behavior." In *Aspects of the Theory of Artificial Intelligence.* C. A. Muses, ed. Plenum Press, New York, 1962. pp. 165–210.

34. PASYNSKIY, A. G. "Some Problems of Biochemical Cybernetics." *Vest. Acad. Nauk USSR. 32:*25–31, 1962. (In Russian).

35. PATTEE, H. H. "On the Origin of Macromolecular Sequences." *Biophys. J.* 1(8): 683–710, 1961.

36. RABIN, M. O., and D. SCOTT. "Finite Automata and Their Decision Problems." *IBM J. Res. and Develop. 3:*114–125, 1959.

37. RASHEVSKY, N. "Mathematical Foundations of General Biology." *Ann. N.Y. Acad. Sci. 96:*1105–1116, 1962.

38. RICE, W. E., and R. M. BOCK. "The Problem of Sequence Determination in Transfer RNA." *J. Theoret. Biol. 4*(3) :260–267, 1963.

39. RICH, A. "The Transfer of Information Between the Nucleic Acids." In *Synthesis of Molecular and Cellular Structures.* D. Rudnick, ed. Ronald Press, Inc., New York, 1961. pp. 3–11.

40. ROSEN, R. "Digital Computers and the Problems of Cellular Regulation." Report read at 1963 Denver Conference of the As-

sociation for Computing Machinery, August 27–30, 1963.

41. SCHUTZENBERGER, M. P. "On the Definition of a Family of Automata." *Inform. and Control. 4*:245–270, 1961.
42. SORKIN, YU. I. "Algorithmic Solvability of the Problem of Isomorphism of Automata." *Doklady Nauk USSR. 137*(4):804–806, 1961. (In Russian).
43. STAHL, W. R. "Solvable and Unsolvable Problems for an Algorithmic Model of a Cell." (Submitted for publication).
44. STAHL, W. R., R. W. COFFIN, and H. E. GOHEEN. "The Turing Machine as a Research Tool for Algorithmic Simulation." (Submitted for publication).
45. STAHL, W. R., and H. E. GOHEEN. "Molecular Algorithms." *J. Theoret. Biol.* 5:266–287, 1963.
46. STAHL, W. R., M. C. WATERS, and R. W. COFFIN. "Pattern Recognition Algorithms Implemented as Turing Programs." (Submitted for publication).
47. SUGITA, M. "Functional Analysis of Chemical Systems in Vivo Using a Logical Circuit Equivalent. II. The Idea of a Molecular Automaton." *J. Theoret. Biol. 4*(2):179–192, 1963.
48. TSETLIN, M. L. "On the Behavior of Finite Automata in Random Media." *Avtomat. i Telemekh. 22*:1345–1354, 1961. (In Russian).
49. TRACHTENBROT, B. A. "Algorithms and the Machine Solution of Problems." (In Russian). Available under the title *Algorithms and Automatic Computing Machines.* D. C. Heath and Co., Boston, Mass., 1963.
50. TURING, A. M. "On Computable Numbers, with an Application to the Entscheidungsproblem." *Proc. London Math. Soc. Ser. 2. 42*:23–265, 1937.
51. TURING, A. M. "The Chemical Basis of Morphogenesis". *Phil. Trans. Royal Soc., Ser. B. 237*:37–72, 1954.
52. TURING, A. M. "Solvable and Unsolvable Problems." *Science News (London). 31*: 7–23, 1954.
53. TURING, A. M. "Can a Machine Think?" Reprinted in *The World of Mathematics.* J. R. Newman, ed. Simon and Schuster, New York, 1956. pp. 2099–2123.
54. ULAM, S. "On Some Mathematical Problems Connected with Patterns of Growth of Figures." In *Mathematical Problems in the Biological Sciences.* Sympos. No. 14 of the Amer. Math. Soc., Providence, R.I., 1961. pp. 215–224.
55. VON NEUMANN, J. "The Genral and Logical Theory of Automata." In *Cerebral Mechanisms and Behavior.* L. A. Jeffress, ed. John Wiley, New York, 1951. pp. 1–41.
56. VON NEUMANN, J. In *The Computer and the Brain.* Yale University Press, New Haven, Conn., 1958. pp. 69–73.
57. WADDINGTON, C. H. In *New Patterns in Genetics and Development.* Columbia University Press, New York, 1962.
58. WANG, H. "A Variant to Turing's Theory of Computing Machines." *J. Assoc. Comp. Mach. 4*(1):63–92, 1961.
59. YOVITS, M. C., and S. CAMERON. *Self-Organizing Systems.* Pergamon Press, New York, 1960.

COMPUTER SIMULATION OF HUMAN INTERACTION IN SMALL GROUPS

John T. and Jeanne E. Gullahorn

Department of Sociology and Anthropology, and
Computer Institute for Social Science Research
Michigan State University
East Lansing, Michigan

and

Consultants, System Development Corporation
Santa Monica, California

The modern digital computer has made major contributions to sociology in at least two general areas. Because of the computer's high speed and its ability to handle effectively the interrelationships among many and complex variables, we are enabled to perform more sophisticated analyses of data already gathered. For the first time it is possible to deal statistically with social systems as systems. Potentially of even greater benefit is the second contribution—the use of computers to generate data for the purposes of theory testing and development.

It has been traditional in sociology to express theory in verbal formulations. Words, of course, are weak tools for achieving precise statements of propositions, for studying the interrelationships among variables, and for learning the consequences for a social system if it operates according to the processes defined or implied by the theory. Once the processes of a theory are expressed as routines in a computer program, we can simulate the behavior of a social system which operates according to the rules of the given theory. To the extent that the observed outcomes of an experiment in a computer run match the observed outcomes of the identical experiment in a real-life system, the rules of the theory may be considered a sufficient (though not a necessary) explanation of the behavior.

Let us turn now to a description of HOMUNCULUS,[3,4] an operating program in Information Processing Language, Version V,[7] designed to simulate human interaction in small groups. The computer model is based on the theory advanced in a recent book, *Social Behavior*, by George C. Homans, which presents an explanation of elementary social behavior—that is, of "face-to-face contact between individuals, in which the reward each gets from the behavior of the others is relatively direct and immediate".[5] Homans' model is one of social exchange, envisaging human behavior as a function of its payoff: in amount and kind, an individual's responses depend on the quantity and quality of reward and punishment his actions elicit. To explain elementary social behavior, Homans advances five propositions, based primarily on experimental psychology and classical economics, dealing with the effects of such factors as the frequency and recency of reinforcement,

stimulus and response generalization, and relative deprivation and satiation.

To simplify the problems of expressing concepts concerning human social behavior in a computer program, we followed the tactic of beginning with a simple interaction sequence between two hypothetical individuals. In *The Dynamics of Bureaucracy*, Peter Blau of the University of Chicago describes in detail a federal civil service office in which the men studied held the same title but varied in competence.[2] As expected, the more skilled workers received more requests for assistance from their co-workers. While the participants in such consultations benefited, they still incurred costs in their exchange. That is, the agent requesting help was usually rewarded by being enabled to do a better job; however, he paid the price of implicitly admitting his inferiority to a colleague who by title was supposedly his equal. The consultant, on the other hand, gained prestige; nevertheless, he incurred the cost of time taken from his own work.

In our computer model of this social exchange we have programmed an interaction sequence beginning with an agent, Ted, emitting a symbol representing a request for assistance to his co-worker, George. The flow chart depicted in Figures 5 through 6 outlines our interpretation of the processes involved in operationalizing Homans' propositions for elementary social behavior. In our illustration, the programmed statement of Homans' propositions specifies the symbol-manipulating processes which enable our hypothetical consultant, George, to decide what action he will emit in response to Ted's request.

As we began to develop flow diagrams for the specific routines to represent each proposition, we found that the discipline of making decisions about how to program a computer to simulate social behavior gave us a different perspective in our thinking about how humans react in social contexts. This new frame of reference forced us to make explicit our conception of a person as an information-processing system. If our model of a person were to behave according to the principles set forth in Homans' explanatory propositions, the format in which he was described in the computer had to be such that he could perform at least the following activities: He had to be able to receive stimuli, recognize stimuli, store stimuli in memory, and compare and contrast stimuli; he had to be able to emit activities, differentiate reward and punishment, associate a stimulus situation with a response, and associate a response with a reinforcement; and, on the basis of past experience, he had to be able to predict the consequences of each contemplated response.

In the group setting he had to be able to discriminate among the members of the group, evaluate each social stimulus in terms of the specific person emitting it, and select a response appropriate to the specific other person in the given group. Our model thus pictures a person as an active hypothesis-testing organism capable of receiving, storing, analyzing, and reconstructing information. These human information-processing functions bear obvious similarity to the digital computer's symbol-manipulating capacities. Even so, the process of constructing such a social being in the computer was no easy task.

Fortunately, IPL-V was designed to handle the types of problems we faced. In our model a person is represented as a list structure composed of a hierarchy of list structures, lists, and description lists. A simplified presentation of the organization of a list structure of a person is depicted in Figures 1, 2, and 3. Among the data included are lists specifying an individual's identity, his general level of ability and his level of skill in social interaction and in certain specific tasks, his relative and absolute position

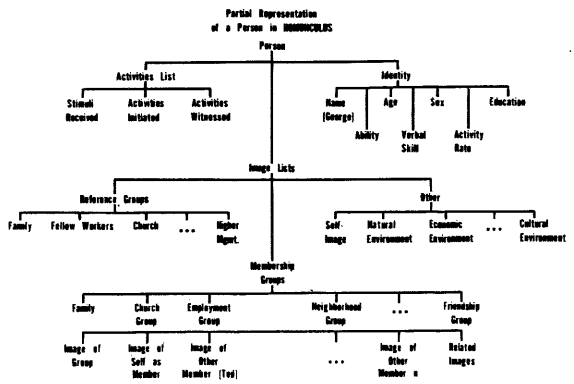


Figure 1.

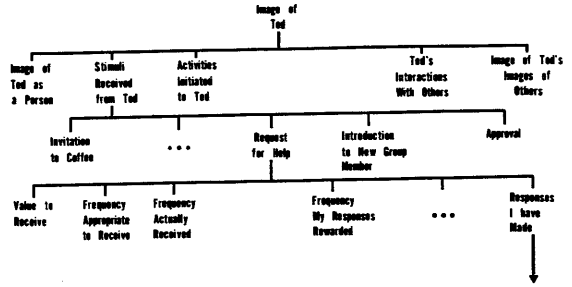


Figure 2.

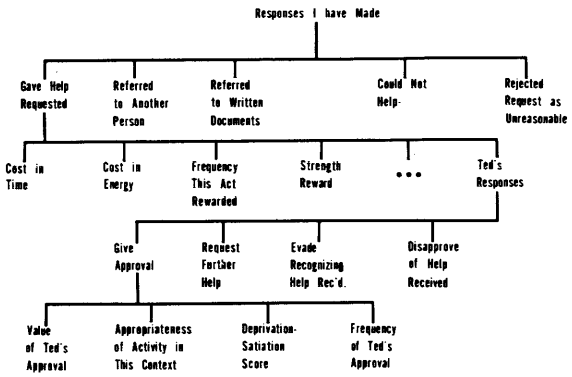


Figure 3.

in certain groups, his images of his membership and reference groups, and lists containing elements of his past history and his resulting values and needs. Through exposure to various experiences an individual's list structure becomes modified to include a record of them. Thus, at the beginning of a run, he may be represented by a list structure of as little as 200 words. By the end of some fifty or one hundred interactions, he may require a thousand or more IPL words to describe him. In effect the individual learns from his experiences and keeps in memory the information that will enable him to behave more appropriately in future interactions.

The processes expressed in Homans' propositions determine the behavior of the person-models in HOMUNCULUS. Homans assumes that his propositions explaining elementary social behavior apply generally to all humans, regardless of their cultural backgrounds and institutional relationships. In operationalizing Homans' propositions, therefore, we designed the routines to be common for all simulated persons. However, the routines are so written that the information retrieved from the data list structures describing individual participants determine whether certain subroutines will be executed and what their outputs will be in specific interaction sequences.

Propostion 1. To illustrate some of the processing involved in our model as well as some of the problems encountered in translating verbal statements into computer routines, let us consider the dynamics of Proposition 1 (P1, Boxes IV through XXII in Figures 4 and 5). Homans states this explanatory proposition as follows:

> If in the recent past the occurrence of a particular stimulus-situation has been the occasion on which a man's activity has been rewarded, then the more similar the present stimulus-situation is to the past one, the more likely he is to emit the activity, or some similar activity, now.[5]

Essentially this describes the effects of the psychological principles of stimulus generalization and response generalization. In formalizing this proposition as a computer routine, however, we found that the apparent clarity of the verbal statement actually cloaked certain ambiguities.

What, for example, is a "particular stimulus-situation?" In terms of the specific interaction sequence we have selected, does it include every request for help one has received, or does it include only requests from the same person for assistance on the identical problem within an identical social context? We decided to write routines to process in sequence two aspects of the "stimulus-situation." That is, we hypoth-

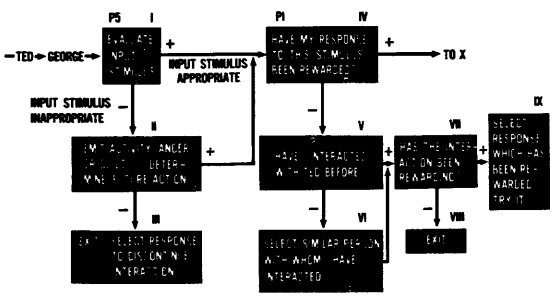


Figure 4.

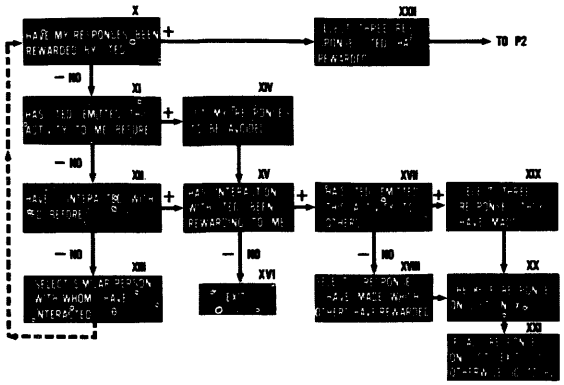


Figure 5.

esized that a person receiving a stimulus, such as a request for assistance, would first react in a relatively global manner to the general situation itself; then he would consider the situation in terms of the specific individual emitting the stimulus. Thus, in our example, we interpreted the initial information processing implied by Proposition 1 (see P1, Box IV in Figure 4) to involve consideration by our office worker, George, of whether the activity received —in this instance, a request for help—is a general stimulus-situation in which his responses have been rewarded. In executing this process, a subroutine representing one of George's information-retrieval capacities searches a memory list of reinforced stimulus-situations to determine whether the present input is among them.

Let us take the positive branch of the flow diagram and thus assume that George has found that in the past he has been reinforced for responding to a request for assistance. His next consideration is outlined in Box X of the flow chart in Figure 5. Now George must determine whether his responses to a request for help have been rewarded by Ted, the person currently introducing the stimulus-situation.

In checking on past interactions with Ted, George must search deeper into his memory structure. At this point one subroutine locates George's image list of office colleagues, finds the sublist describing Ted, determines whether Ted has emitted the present stimulus to George previously and if so whether Ted has generally rewarded George's responses to such requests. If he thus determines that Ted has been an agent of reinforcement in past interactions involving requests for assistance, then George proceeds to retrieve the types of responses he has emitted to Ted. In our program George selects up to three activities from a memory list of responses Ted has rewarded (Box XXII, Figure 5); then he proceeds to process further information regarding these contemplated actions in order to select a response which he anticipates will net him a social profit.

It is apparent that we have adopted as a tactic the policy of including, as a group, stimuli which resemble each other but are not necessarily identical. This means, however, that our programmed people cannot discriminate degrees of similarity. No great difficulties arise, of course, in constructing routines to test gradations of similarity as called for by Homans' proposition. When we do include a test of similarity, it will require that we be more precise than Homans in specifying what we mean by "more" and by "similar." To develop this discrimination we plan to build a list of attributes describing various activities. One such attribute would have scaled values indicating the expenditure of energy required to emit an activity; another would have a range of values denoting the amount of time required to complete the given activity. Once these description lists for activities have been completed, a routine can be written to compare activities, determining the number of attributes two activities have in common as well as the degree of similarity in the values of their attributes. The criteria for judgment of similarity can be set as fine or as broad as desired. Our interest will

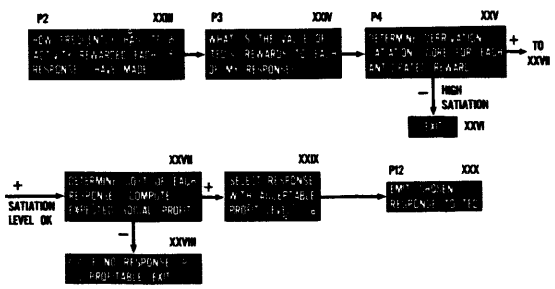


Figure 6.

be the development of routines which simulate human decision processes as closely as possible.

Up to this point in our discussion of Proposition 1 we have considered only the positive branches—that is, what happens if George finds that the general stimulus-situation has been a rewarding occasion and that Ted has been an agent of reinforcement. But one may note that most of the items included in the flow diagram in Figures 4 and 5 outline processes accommodating the negative branches. For example, if George finds that in the past his responses to a request for help generally have not been reinforced (Box IV, Figure 4), he would still wish to ascertain whether he has interacted with Ted before (Box V) and if so, whether such interactions have proved rewarding (Box VII). If past interactions with Ted have had positive consequences, George may then select a response Ted has previously reinforced to try in this stimulus-situation (Box IX). Otherwise, George may decide to withdraw from what is likely to be another unrewarding situation (Box VIII). Even if he has not interacted with Ted previously (Box V), if George is a gregarious, other-directed American, he may still wish to continue the interaction; therefore, he can select some person similar to Ted (in terms of such attributes as sex, age, relative position in the group, etc.) to use as a reference source for determining appropriate action (Box VI).

Let us now consider the processing involved in the negative branch for the inquiry concerning the social context of the stimulus-situation. To be at this point in the program George would have ascertained that, in the past, responding to a request for help had generally proved rewarding (Box IV, Figure 4); however, he would have found, in considering the request in terms of the current stimulus person, that his responses had not been rewarded by Ted (Box X, Figure 5). A relevant question then is whether George previously has received this stimulus from Ted (Box XI). If he has not, then George will ascertain whether he really has interacted with Ted before (Box XII); if not, he will select a colleague similar to Ted to use as a reference source in processing Ted's request (Box XIII). To indicate the possibility that George is using a surrogate for the stimulus person in making decisions, references to Ted appear in parentheses.

If George finds that Ted has previously requested his assistance (Box XI), then the responses he made to Ted's request should be avoided since they did not lead to reinforcement (Box XIV). Indeed, in such a case George will wish to evaluate his relationship with Ted to determine whether interacting with him is worthwhile (Box XV). If it is not, he may end the interaction sequence then and there (Box XVI). On the other hand, if George has enjoyed contact with Ted (Box XV), he might try to recall whether he has observed Ted's asking other co-workers for assistance (Box XVII) and, if so, what their reactions were (Box XIX). If he lacks this knowledge of group activities (Box XVII), George still can select rewarded responses to requests for help from some other colleagues (Box XVIII). In either case, George will then ascertain whether the responses suggested are of the same type as those he emitted previously without receiving reinforcement from Ted (Box XX). If they are, George has little choice but to withdraw from what would probably be an unrewarding situation; otherwise, he can try again to elicit a positive reaction from Ted by emitting one of the untried responses (Box XXI).

Proposition 2. Homans' second proposition deals with the positive influence of the frequency and recency of reinforcement:

> The more often within a given period of time a man's activity rewards the activity of another, the more often the other will emit the activity.[5]

Reformulating this proposition for computer simulation posed a number of problems. It would have been relatively simple merely to set a counter for each reinforced response and then retrieve the desired information regarding reward frequency. However, we felt this procedure would not adequately simulate human information-processing systems. Of course, people do avail themselves of precise measurement scales and use various cultural artifacts—such as computers—to increase their accuracy. But in making estimates concerning frequencies and values of rewards ensuing from everyday social interaction, people seem to use a less re-

fined means of measurement. In programming this proposition, therefore, we devised a rather crude five-point ordinal scale for reward frequency, ranging from an estimate that a response was "nearly always rewarded," through a judgment that it was "rewarded about half the time," to an assessment that was "almost never rewarded."

At present we are experimenting with different means of manipulating this scale. One routine we have written increases the ordinal scale value for the reward frequency after three reinforcements of the response. This procedure, however, is not completely satisfactory. Indeed, one may argue that estimates of reward frequency are not necessarily independent of the emotional salience of the reinforcement. When HOMUNCULUS has reached the stage of simulating small-group behavior in controlled conditions, it should be possible to test various approximations of human judgments of reward frequencies from social interaction and to select the routines which simulate the actual behavior most accurately.

When the processing for this proposition is completed (P2, Box XXIII, Figure 6), George has a rough estimate of the frequency with which Ted has rewarded each of the activities he is considering in response to Ted's current request for help. Homans' Proposition 2, taken alone, would lead to the expectation that George would then merely emit the most frequently rewarded response alternative. But other information must be processed before a decision is reached.

Perhaps here we should indicate how the program keeps all this material in immediate memory for George. Up to one hundred named private storage cells are assigned for this purpose, and instructions in each routine specify which cells it is to use for storing its findings. At present, George is using about fifty of these cells. In addition, important information available to all group participants—for example, what could be seen and heard during the last five interactions—is kept in named public storage cells.

Proposition 3. Among the other relevant factors that must be considered in selecting an activity to emit is the value of the anticipated reward. Homan's third general proposition states,

> The more valuable to a man a unit of the activity another gives him, the more often he will emit activity rewarded by the activity of the other.[5]

Assessing the value of an activity is somewhat more complicated than estimating the frequency with which it occurs. Value has two components—one relatively constant, and the other, which we shall discuss in Proposition 4, relatively variable for the periods of time involved in the simple interactions comprising elementary social behavior. The value component referred to in Proposition 3 concerns an individual's rank-ordering of the subjective reward attendant on receiving one activity rather than another. With reference to our example, we might predict that George would find warm social approval involving Ted's complimenting him in front of colleagues to be more "valuable" than a half-hearted response of "Hmm, thanks," or an annoyed retort, "Well, sorry I bothered you."

At this point in our program, therefore, we have what game programmers term a "look-ahead." In considering Ted's request, George has "in mind" three responses he recalls Ted's having rewarded in the past, and he has estimated the frequency with which Ted has reinforced each response. Now he must consider more carefully the particular reward he expects Ted to give to each response so that he may determine the inherent worth of each anticipated reward (P3, Box XXIV, Figure 6). Taking in turn each activity George is contemplating, the routines executing this proposition retrieve the responses Ted has previously made to each, determine which one he is likely to emit now, and search description lists to find the subjective value of the reward for George.

Proposition 4. Homans' fourth proposition deals with the other component of value—the deprivation-satiation aspect, or the marginal utility of a given unit of activity.

> The more often a man has in the recent past received a rewarding activity from another, the less valuable any further unit of that activity becomes to him.[5]

In contrast to the relatively constant intrinsic satisfaction aspect of value, the deprivation-satiation component varies over a range of possible rankings. Taking into account the amount of an activity a person has received, we note that he "values" that activity more when he has been deprived of it than he does when he is in a state of relative gratification. Thus, while social approval may be highly rewarding to an individual, if in the recent past he has received a great deal of this generalized reinforcer, then he is not likely to be so interested at the moment in receiving more.

In processing the information necessary for completion of this stage of the program, George must evaluate his relative deprivation with reference to the rewards he anticipates from Ted. George now has in immediately available memory a record of each activity he is contemplating and stored with each activity is various information about it, including the response he expects Ted to make to it. The routines which execute Proposition 4 search the description lists of each of the anticipated rewards to determine the degree of George's current deprivation or satiation with respect to them. A deprivation-satiation score based on a simple ordinal scale is stored as the value of a special attribute on the description list of each activity. In executing Proposition 5, which we shall discuss later, routines update the deprivation-satiation score whenever an activity is received.

With the information retrieved thus far George has an estimate of the relative frequency with which Ted has rewarded each activity he is considering emitting. Furthermore, he has predicted Ted's reaction to each of the projected actions and has determined how rewarding each of these anticipated reactions is to him, personally, as well as how deprived or satiated he currently feels with respect to each of these expected rewards. At this point, therefore, George can rank his contemplated responses in terms of their expected payoff. But he is not yet ready to emit the highest ranked action.

Another important consideration is the cost of the proposed response. Homans defines the cost of an activity as the value of the reward obtainable through an alternative activity foregone in emitting the given one. In our example, George must forego working on his own assignment if he takes time to assist Ted; therefore George must determine the relative reward value of this alternative activity. To do this he follows a procedure analogous to that just described, processing information concerning the frequency of past reinforcement and the value of the anticipated reward ensuing from this activity as well as his relative satiation with the reward. Then he can compare the overall expected reward from his contemplated response to Ted with the anticipated reward from continuing with his own work, and he can compute what Homans terms the psychic profit—the reward of an activity less its cost.

Let us suppose George is tentatively planning to give Ted direct assistance on his problem because in the past Ted has praised him for this activity, and social approval is a reinforcement George values highly and one for which he feels relative deprivation at present. But let us also suppose that George has an important assignment to complete, and that taking time from it might detract from the quality of his work and thus lessen the approval he anticipates from his boss for a good job. In this case George would incur a loss rather than a profit in helping Ted directly; therefore, he will continue processing to see whether one of the other activities he was contemplating might yield a profit. In this illustration, George will probably decide that referring Ted to another source will net him a profit, since he expects some approval for this activity (albeit less than he would get from directly assisting Ted), and he will incur a very small cost in terms of time taken from his own work.

Having selected what he expects to be a socially profitable activity, George emits that response to Ted. At this point our program cycles, and the activity George has emitted becomes the activity Ted has received. Now Ted must process information in order to select an appropriate and profitable response to George.

Proposition 5. Distributive justice, the subject of Homans' fifth proposition, is perhaps the most complex of the concepts involved in the explanation of elementary social behavior. At the very least it requires consideration of

information at another level—that of social norms or accepted expectations for behavior within a group. Through repetition of interaction situations within a group, certain behavior patterns become stabilized so that expectations develop regarding what constitutes justice in the distribution of rewards and costs between persons: The greater a man's costs in a given interaction, the greater his rewards ought to be But the implications of distributive justice go even further, taking into account a person's investments in an interaction—for example, his seniority, skill, experience, age, and sex. The greater the investment a person makes in an interaction, the greater the net profit he has a right to expect. Thus, according to the principle of distributive justice it is consensually expected that certain antecedent costs and investments should have as consequents certain types and degrees of reinforcement. Homans states the related proposition as follows:

> The more to a man's disadvantage the rule of distributive justice fails of realization, the more likely he is to display the emotional behavior we call anger.[5]

More is included, however, for if a man receives rewards beyond those to which he considers himself entitled, he is likely to experience guilt feelings.

Translating this proposition into computer routines posed some of the most interesting problems we have yet encountered in working with HOMUNCULUS. In effect, the list structures of our agents had to be programmed to have consciences, and they had to include a repertoire of appropriate anger responses.

In essence, our programmed interpretation of this proposition asks whether a stimulus activity is appropriate in the given circumstances (P5, Box I, Figure 1). If so, then the person receiving it can process it as George did Ted's request, which he considered appropriate. If, however, the stimulus activity is judged inappropriate, then more complex behavior results. To illustrate this, let us shift to a description of the interactions between George and Tom, another worker in the same agency.

It is an accepted office norm that a worker who asks for help should do so openly in a manner acknowledging the superiority of his consultant with respect to the given problem. Tom, however, has been seeking aid from George in a rather devious manner, coming to George with "an interesting problem" and saying he would like to see whether George arrives at the same solution as he. This has occurred three times in the recent past, and on each occasion, Tom has greeted George's suggested solution with the comment, "Yes, you reached the same conclusions I did." George decides Tom is violating the norms of fair exchange by evading the cost of thanking him for his assistance and conceding his superiority. The fourth time Tom presents him with an interesting problem George angrily responds, "Look, why don't you do your own work!"

This description, of course, does not answer the question of how the computer is programmed to behave in such an all-too-human way. George is programmed to treat time spent solving a problem presented by another worker as being help to that person for which recognition and social approval are due. When his colleague responds to his efforts with an unrewarding confirmation that he arrived at the same conclusion, George finds this input inappropriate in terms of his expectations regarding distributive justice. Therefore, routines processing Proposition 5 change George's image list of Tom so that next time he expects greater recognition and thanks than normal to atone for the present evasion. After three repetitions of this interaction sequence the discrepancy between Tom's behavior and George's expectations will be so great that when George evaluates Tom's response he will plant a signal in his image list of Tom to indicate that interacting with him is not rewarding because Tom violates group norms.

The next time Tom asks for an opinion after this warning signal has been set, George will respond by displaying anger or by storing up aggression to be expressed against someone else. In the computer program an anger response involves emitting behavior punitive to another person. But before actively punishing Tom, George will first assess the consequences to himself of such behavior. In one possible interaction sequence, if George finds that Tom is in favor with George's own boss, he may sup-

press his aggression at the moment and then release it the next time he interacts with a subordinate.

The routines processing the negative branch of Propostion 5 (Box II, Figure 4) thus not only modify image lists but also use some of the routines from the other propositions to evaluate the probable consequences of direct anger responses. Depending on the outcome of this processing, the program either proceeds to Proposition 1 or the interaction is terminated.

Once HOMUNCULUS was operating effectively—so that it simulated behavior characteristic of the civil service office discussed earlier—we faced the problem of how the model could be used to provide a more general test of Homans' theory and also to simulate across a broader range of experiments involving both individual decision-making and group interaction. Actually, the model as first completed could have been used; however, the task of building a complete repertoire of activities appeared overwhelming, especially when we confronted the need to provide to each group participant with a complete history of activities received from and emitted to each of the others plus the values and costs of each activity in each situation.

In our new program of activities we have adopted a scheme of twelve categories which appear to encompass all problem-solving activities.[1] These categories include six dimensions of group interaction: 1) communication problems where an individual either gives orientation (information, repetition, clarification, and confirmation) or asks for orientation; 2) evaluation problems where an individual either offers an opinion (evaluation, analysis, and expression of feeling) or asks for an opinion; 3) control problems where an individual either presents a suggestion (direction and possible ways of action) or asks for a suggestion; 4) decision problems, where an individual either agrees (shows passive acceptance, understands, concurs, and complies) or disagrees; 5) tension reduction problems where an individual either releases tension (jokes, laughs, and is satisfied) or displays tension; and 6) reintegration problems where an individual either shows solidarity (by raising the other's status and by giving help and reward) or shows antagonism.

On the basis of laboratory studies of small groups we know the approximate frequency of each type of activity, the changes in relative frequency of each type of activity through time as a group interacts, and the types of responses likely to occur to each type of activity. This information is programmed into reference groups, and data such as the appropriateness of a given response to an activity differ for various reference groups—e.g., for management, as contrasted to a labor union. When the individual participant does not have in his own repertoire sufficient information to select a response to a stimulus, he checks with his reference group to find one to consider.

As the program is now written we no longer specify the first actor and action. Currently we are running interactions among triads. When the program begins operating, control is turned over to the master executive routine. This checks to determine whether a member of the triad needs to act. If so, control is turned over to the actor. If not, the need for interaction of each member is incremented by a figure determined by his basic need for social interaction plus his energy level. The program cycles until some member passes the threshold level, and then he assumes control.

If the initiator knows one or both of the other members of the triad he selects the one with whom to open interaction on the basis of expectations for securing a rewarding response. If the group is composed of unacquainted members, the initiator evaluates the other two and selects the one whose characteristics lead to a prediction of rewarding interaction. This involves assessing the "personas" or public images of the other members and comparing their characteristics with his own self-image to determine compatibility.

If the initiator knows the person he has selected, he usually chooses an activity to emit on the basis of past experience. Otherwise, he consults his primary reference group to find an activity likely to lead to positive reinforcement when emitted to a person like the one chosen—that is, to an individual who is, say, more skilled in the task than he, less skilled in social interac-

tion, one level above him in social background, and an adherent of similar reference groups.

Interaction thus begins and proceeds according to the flow diagram discussed earlier. The propositions guiding behavior remain the same; however, because of refinements introduced—for example, in the routines enabling one member to interrupt another before he has opportunity to emit a chosen activity—the process has become more complex. A programmed member may keep in available memory at least two sorts of activities he may wish to emit: a normally selected response which was prevented by someone else's interruption, or an anger response which the individual suppressed because of the given circumstances. The need to emit such thwarted responses fades as other activities intervene, so that after five activities have been emitted by others, the individual "forgets" having been interrupted and goes on to select other activities. However, the mere fact of being interrupted raises his need to participate, and increases the likeihood of his interrupting in turn to carry out his previously selected activity or to shift his attention to other activities.

Discontinuity and annoyance responses are likely to arise when activities are focused on social-emotional areas and when a series of interruptions occur. When the focus is on task areas, interruptions may lead to annoyance responses, but the discontinuity usually does not appear because the activities continue within the context defined by the given task. Memory of anger responses does not deteriorate so rapidly, but even this disappears after a series of twenty activities.

We mentioned that at present we are running experiments dealing with three-person groups and discussed the processes through which interaction within the group is initiated. Some of the findings in this area have proved to be of theoretical interest. For instance, a group of three unacquainted persons usually breaks down into a pair and an isolate. On the other hand, a group of three already acquainted persons remains a group of three if all the pair relationships are positive. If the pair relationships are not all positive, the outcome is more complex and may range from a relatively unstable group of three to three isolates who never become a group.

Outcomes from such interactions using HOMUNCULUS help explain what happens. Sooner or later a member of the unacquainted group initiates interaction with one of the others. Each time this pair's interactions prove rewarding, information is stored in their image lists which lessens the cost of finding rewarding acts to emit to each other and increases the expectation of a rewarding response. Thus the rate of interaction between them increases, the repertoire of rewarding activities increases, and the likelihood of the bond's being disrupted lessens. The third man does not have the opportunity to let the other two know how rewarding he might be and remains an enigma. Thus the interaction leads to a dyad and an isolate.

But if the first interaction is punishing to one or both of the two involved, the probability becomes higher that interaction will be initiated with the third person. Assuming that his values and norms make rewarding interaction likely, then whichever one captures him first will, in most instances, form a dyad with him and leave the slower moving man as isolate. If no rewarding pair is found, then in most circumstances a group never forms.

With a group of three persons who already know and like each other, the same principles lead to different outcomes. Each has already established a repertoire of activities that he knows he can emit to the other two and that will be rewarded. Therefore, when there is slight satiation with the rewards one of them can provide, attention is temporarily transferred to the other. Occasionally, also, the interrupt mechanism leads the one not yet active in the group to join the conversation. Since the others have experienced rewarding interactions with him, he is not rebuffed.

In reaching beyond the simulation of small-group laboratory experiments, we have begun planning experiments to use HOMUNCULUS to test theories of organization design. None of these will be programmed and on the machine for at least a year, and probably longer. This appears to us to be a particularly promising area in which to blend contributions to sociological theory and to organizational practices.

As our first study in this area we plan to examine the relative effectiveness of the traditional authoritarian hierarchy *vs.* the multiple-overlapping-group design advocated by Rensis Likert.

In the traditional structure, communication occurs between adjacent men up and down the line, with information transmitted up, decisions made at the top, and instructions flowing down the line. Theoretically, lateral communication does not occur. In the overlapping-group organization, each member within a group is in regular communication with every other member; decisions are reached by the group rather than the superordinate; and the members are responsible for passing on the information to other groups of which they are members.

Computer simulation of decision-making activities by both types of organization is expected to produce information showing the relative efficiency of the two designs. Such factors are considered as the amount of information available to those making decisions; the effect of the decision-making process on the motivation of the workers to perform; and the impact of each type structure on the satisfaction of individual workers, on their sentiments toward each other, and on their sentiments toward the organization.

CONCLUSION

In our discussion of HOMUNCULUS as an application of computer simulation to problems of sociological or social-psychological theory we have tried to demonstrate that research involving computer models not only plays the passive role of testing and verifying theory; it performs active functions for the development of theory. Formulating a theory as a computer model affords one a relatively tractable representation and possibly a more meaningful conceptualization, having increased precision as a result of the clarification of concepts the programming process necessitates. This organizational function of the model is further enhanced by the computer's capacity actually to set the theoretical processes in motion and output data generated as logical consequences of hypothesized processes. While investigating the extended logical consequences of a theoretical system is in itself a rewarding activity for a social theorist, another benefit involves the possibility of generating unanticipated findings leading to refinement of theoretical constructs or elaboration of a theory and to experimentation directed to the verification of new hypotheses. This possible serendipity bonus in addition to the increased precision, tractability, and dynamic capacity of a computer model gives the social theorist ample reward for the cost of translation from verbal formulation.

REFERENCES

1. BALES, ROBERT F. *Interaction Process Analysis: A Method for the Study of Small Groups.* Cambridge, Mass.: Addison-Wesley Co., Inc., 1950.
2. BLAU, PETER M. *The Dynamics of Bureaucracy,* Chicago; University of Chicago Press, 1955.
3. GULLAHORN, JOHN T., and GULLAHORN, JEANNE E. "The Computer as a Tool for Theory Development." Santa Monica, California: System Development Corporation, SP-817, June 5, 1962. To appear in Dell Hymes, editor, *Uses of Computers in Anthropology.* The Hague, Netherlands: Mouton and Company, in press.
4. ————. "A Computer Model of Elementary Social Behavior," in Edward Feigenbaum and Julian Feldman, editors, *Computers and Thought.* New York:McGraw-Hill, 1963. Also reprinted in *Behavioral Science,* October, 1963, 8, No. 4, pp. 354-362.
5. HOMANS, GEORGE C. *Social Behavior: Its Elementary Forms.* New York: Harcourt, Brace & World, 1961, pp. 7, 53, 54, 55, 75.
6. LIKERT, RENSIS. *New Patterns of Management,* New York: McGraw-Hill, 1961.
7. NEWELL, ALLEN (Ed.). *Information Processing Language—V Manual,* Englewood Cliffs, New Jersey: Prentice-Hall, 1961.

REAL-TIME COMPUTER STUDIES OF BARGAINING BEHAVIOR: THE EFFECTS OF THREAT UPON BARGAINING*

Robert J. Meeker, Gerald H. Shure, and William H. Moore, Jr.

System Development Corporation
Santa Monica, California

INTRODUCTION

Most behavioral scientists are well aware of computer technology as a means of reducing and otherwise analyzing empirically derived data. A small number have also discovered, and enthusiastically endorsed, the computer's potential for simulation of analytic models. But little has been made of yet another application of computer technology in the behavioral sciences, namely, the use of computers for data acquisition.

To anyone familiar with the development of real-time data processing, the prospect of a computer-administered experiment suggests a general increase in efficiency and flexibility over the manual mode of experimentation. One tends to think immediately of greater speed, greater accuracy, and, in some instances, greater economy for a research program; but we are emphasizing the fact that certain types of behavioral studies can be greatly broadened and enhanced by exploiting the computer's efficiency and ability to *gather* the desired data. In this paper we shall develop the thesis that computer-administered experimentation permits us to gather an important type of behavioral data which are now practically unobtainable; as evidence we shall cite our computer-based empirical research on bargaining and negotiation behavior.

Description of the Project

Our research is designed to study bargaining and negotiation behavior, particularly the dynamic interaction process that takes place in mixed-motive (non-zero sum) bargaining situations. This study utilizes the computer for real-time experimental control and assessment of subjects at critical points during bargaining situations. The project goal was to develop a general computer-based experimental vehicle which would provide unique opportunities to study bargaining and negotiation behavior and relate it to social-psychological factors (e.g., threat, trust, cooperation, status, power), personality variables, and game-strategic characteristics and tactical moves.

General Problem Background

Bargaining and negotiation behavior is a striking example of an area of study which can be broadened by having access to data that have heretofore been elusive. Investigations in this area have typically been confined to data on the overt behavior of the bargainers (i.e., the bids, moves, outcomes, rates of response, and so on); yet investigators, theoreticians, and bargainers themselves recognize that overt behavior is directly related to subjective attitudes, intentions, expectations, and perceptions. Clearly, in the temporal sequence of bargaining exchanges,

* This research was supported by the Advanced Research Projects Agency, under contract SD-97.

participants shift their evaluation of outcomes, alter their goals, and attempt to induce their opponents to alter theirs. By these criteria, a complete analysis of bargaining requires not only data on the overt pattern and sequence of verbal exchanges, moves and countermoves, but also a parallel assessment of the participants' changing intentions, expectations, and perceptions of relative status and opponent behavior.

Consequently, we must ask the bargainer what he is doing, or thinks he is doing, what he plans to do, and why. The importance of these kinds of "subjective" data is, regrettably, matched by the difficulties of collecting and interpreting them. To obtain useful answers from a subject, questions must be pertinent to what has been going on, unambiguously stated, and not be unduly suggestive. Unfortunately, it is not yet economically feasible to put each experimental subject on a psychoanalyst's couch. But even if it were feasible, the highly personalized interrogation creates additional problems of its own—those of comparability of questions and answers and the effect of the presence of the interrogator on the responses made by the subject. Small wonder that psychologists have developed strong objections to methods for obtaining subjective data during the ongoing experiment. These objections, economic and methodological, have virtually excluded such data even from studies for which they are most pertinent.

Another important aspect of bargaining and negotiation is its dynamic nature. Two free-acting agents may effect an endless variety of interresponse patterns. Such dynamic interaction phenomena are important for the understanding of behavior in a variety of social-cognitive situations; yet the traditional methods for recording and analysis of social interaction data are extremely costly and place exorbitant time demands on highly trained professional personnel.

In our studies, the computer is used to overcome these difficulties by mediating and monitoring the communications between experimental subjects. Since all subject interactions are conducted via the computer, it is easier to oversee, record, analyze, and even control the interaction process. Additionally, the computer capacity for rapid recording and analysis is used for constant monitoring of subject behavior; and this, in turn, becomes the basis for selective interrogation, by the computer, of subjects on their individual intentions, perceptions, motives, and expectations. Thus, the computer serves the unique function of a standardized interviewer which carefully, but unobtrusively, monitors all participating subjects, detects all situations about which further information is desired, and then selectively asks questions which are relevant to what has just happened to the subject. The impersonal atmosphere eliminates in some measure the problems associated with the presence of the examiner and would appear to encourage freer responses.

Definition of the Bargaining Situation and the Possible Effects of Threat

Three essential characteristics of a bargaining situation may be identified: (1) For both parties there are a number of possible agreements and each party would be better off, or no worse off, in reaching an agreement than in failing to do so. In game theory terms this means that bargaining is involved with variable-sum games, not constant-sum games. That is, the sum of payoffs is not fixed—more for one player does not necessarily mean less for the other. (2) For each party there is a range (at least two) of such potential agreements. (3) Each party has preferences or interests with regard to the possible agreements that are generally opposite to those held by the other party; at the very least both players cannot be maximally satisfied at the same time.

Bargaining is thus a situation in which the participants have mixed motives towards one another. It is neither purely coordinative or cooperative, nor purely competitive. That is, the ability of one participant to gain his ends is dependent to an important degree on the choice of decisions that the other participant will make.

The bargaining processing itself may be either *explicit* or *tacit*. Typically, when we think of bargaining, we think of the explicit mode of direct verbal exchange—the dialogue of bids, counterbids, concessions and compromise until an agreement is reached. On the

other hand, one can bargain in situations where there is little or no opportunity (or no desire) for direct verbal exchange. When drivers might meet at an intersection, or competitors engage in a price war, their dialogue may be limited to the actions that they take. Our experimental situation focuses on this tacit form of bargaining.

Bargaining situations may also involve threats—a means of directly signalling and/or imposing loss or damage on the other bargainer. Such means might entail mutual loss as in a strike or an extortion attempt. Speculation on the effects of threat on bargaining runs in two directions. One position tends to underscore the deleterious effects of threat. The threat spiraling theory predicts a pattern of escalation which leads to disrupting hostility, claiming that perceived threats (whether intended as such or not) will beget counterthreats. In this framework, the mere possession of a threat is sufficient to initiate the escalation process.

In contrast, one finds among other writers an almost sanguine acceptance of coercive and threatening techniques as bargaining tools. In this view, since every bargaining situation involves some minimal aspects of threat—at the very least a bargainer can refuse to reach any agreement—threats present themselves as natural and necessary means of bargaining; means, which can and will be used to effectively define and convey importance of an outcome, degree of resolve, or various other facets of the bargaining relationship.

One finds then what appears to be, on the surface at least, inconsistent expectations regarding the employment of threat. To the advocate of its use in bargaining situations, coercion is specifically designed to influence the "rational decisions" of the other party by controlling his expectations of outcomes which will follow from his actions. Contrasted to this, we find descriptions of threat-evoked behavior which is anything but rational and which leads to consequences desired by none. Since neither party to this argument has clearly specified the conditions which bound the range of his generalizations, the extent of the disagreement is not apparent. Since both views cannot simultaneously hold for the same situation, it becomes important to determine empirically the bargaining conditions under which threat will have either of these effects.

Laboratory Studies on the Effects of Threat

In our investigations we have focused on the effects of threat on *interpersonal* bargaining. In this regard we were particularly interested in the widely cited findings of a series of experiments by Deutsch and Krauss.[2, 3]

In their experiments a bargaining game was played under three conditions: (1) In the bilateral threat condition, both players could take an action which directly imposed an obstacle to the other's making money. (2) In a unilateral threat condition, only one of the players had the means of imposing the obstacle. (3) In a third condition, neither individual had a direct threat capability.

The following results were obtained: Where neither subject possessed a threat capability, the profits increased over trials and were positive for both players. Where one of the players possessed the threat capability, the profits were negative in the early trials, but improved and became positive for both players. In the bilateral threat condition, there was no improvement over trials and earnings remained negative throughout—that is, both players lost money instead of making money.

From the results of their bargaining game, Deutsch and Krauss derive a three-step process occurring in threat-counterthreat spiraling: (1) If there is a conflict of interests, and threats are available, they are likely to be used. (2) The threatened person, if he feels superior or equal in status to the threatener, will feel hostile and will use counterthreat or show increased resistance to yielding. (3) As a result, further intensification of the competitive interest of the bargainers occurs and reduces the likelihood of their arriving at a cooperative agreement.

We should note that this hypothesis depends on references to subjective attitudes, expectations, intentions and perceptions. Reviewing their experiment, we speculated about other hypotheses—equally dependent on intentions and perceptions—which might also account for these results. Our reservations would not have been satisfied by a simple straightforward repli-

cation of their experiments, for even supposing an exact replication of results down to the level of moves and counter-moves, we should still want to know whether the attendant pattern of intentions and perceptions support their hypotheses regarding the perception of threat and the escalation of hostility. Indeed, this is precisely the sort of question for which we can acquire direct data through the use of computer-administered experimentation.

To illustrate how these capabilities can be implemented and used, we turn to a consideration of our experimental vehicle and the details of our investigations on the effects of threats in bargaining behavior.

General Experimental Gaming Vehicle

Experiments were carried out in the Systems Simulation Research Laboratory of the System Development Corporation. The equipment configuration included the Philco 2000 computer, computer-tied 8-by-11-inch television consoles, and associated switch insertion response consoles. Computer programs pair as many as 24 subjects who then play against each other. Subjects can be considered "on line" with the computer. They send messages (moves, bids, threats, offers) to their paired opponent via switch insertions which are displayed on the receiver's TV console. Computer programs present the game situations to the subjects, assist in umpiring of legal moves, provide displays of game-relevant information, record all moves, messages and times of response, and probe the subject's bases for his actions.

Procedures of Experimentation

Our experimental situation incorporated the formal features of the Deutsch and Krauss experiment, although the simulated situation was different. In our Communication Game, subjects were told that they were going to perform as operators in a communication system. In each trial, the subject's task was to transmit messages from a set of one six-letter and two four-letter words. Each subject was informed that he was assigned to a communication channel which would also be used by one other subject. This channel, common to a pair of subjects, had a total storage capacity of six units. The six-unit channel storage limitation prevented concurrent transmission of messages

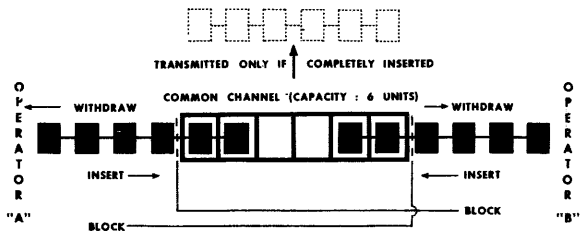


Figure 1. Communications Game.

by both subjects. Figure 1 illustrates the game moves and relationship between the bargainers.

An individual subject could earn 25 cents for transmitting a six-unit message, ten cents for each four-unit message, or nothing if he failed to transmit any complete message. Transmission rules resricted an individual operator to earning a maximum of 25 cents and a pair of subjects to joint earnings of 35 cents per work period.

Work periods were made up of 15 "joint-action" turns during which both subjects would take their actions. In each turn, a subject chose from one of three action alternatives: inserting a letter unit into channel storage, withdrawing a letter unit from channel storage, or passing (taking no action). A letter unit could be inserted successfully only if there were sufficient space in storage (i.e., the six-unit storage capacity could not be exceeded). If attempted insertion(s), together with the units previously in storage, exceeded the channel capacity of six units, then an "overload" resulted, and this indication was displayed as the reason that the attempted insertion(s) had not entered channel storage. In effect, subjects had to work out an arrangement for sharing the channel.

In the "threat" conditions, both subjects were provided with an additional "block" action which prevented the other operator's attempted insertions from *entering* channel storage. The effect of the block action was enduring (it did not require reinstatement to remain in effect). Both subjects could independently take block actions on any turn and maintain them for as long as they wished, or remove their own blocks, but each subject could initiate the block only once in any given work period.

Subjects received instructions in groups ranging in size from 18 to 24 members, were as-

signed to individual cubicles, and did not know with whom they had been paired. Each subject had a switch insertion box for taking actions and a television receiver for the display of channel status and the results of their attempted insertions. Subjects had no other means of direct communication with one another.

All subjects were explicitly told to make as much money as possible regardless of how much the other operator made. After each trial they were informed what they each had transmitted. There was real money at stake in all but one condition.

The computer program permitted each pair to proceed at its own rate of response. Following each work period, individualized computer-designated questions were displayed to the subjects to which they responded by means of their switch insertion boxes. The questions presented were contingent upon moves taken and the results of the immediately preceding work period, and focused upon perceived significance of these events.

Seven experimental conditions were studied. Six of these were variations of the "threat" condition involving bilateral availability of the block. In all of the threat conditions, the action, if maintained, had the direct effect of preventing the other bargainer from making any money. The seventh was a "no threat" condition in which neither operator possessed a block action with which to control the other operator's behavior. Since the block conditions did not differ significantly from one another, they have been combined for the present discussion and compared with the results obtained for the "no threat" condition.

Results and Discussion

Having mirrored the salient features of the Deutsch and Krauss game, we may determine whether any of the steps described by Deutsch and Krauss are found in our data. Let us look first at the six conditions where threat action (the block) was available.

If threat is available, will it be used? The answer is a resounding "yes." Ninety-three per cent of the *194* subjects use it on one or more occasions during the 20 trials. Furthermore, there is little hesitation in employing it. The median trial of initial use is the first trial. (See Figure 2.)

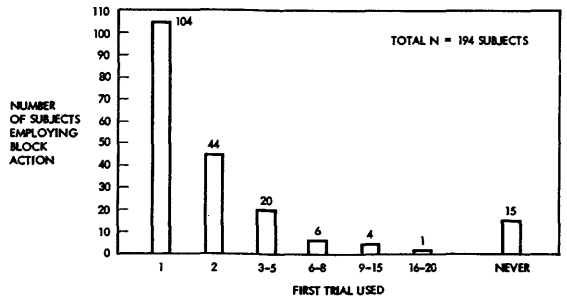


Figure 2. Trial of First Block Action.

Will the threat evoke counterthreat? If one subject employs the block, does it evoke reciprocation? Again the answer would appear to be clearly affirmative. Of the *95* pairs where the block was employed, only 11 subjects never employed the block in return. Furthermore, the median latency of counter response is short, with two-thirds of the subjects responding on the same or immediately following trial. (See Figure 3.)

Continuing to look exclusively at those experimental conditions in which the block action is available, we compared the bargaining outcome for those pairs where both members use the threat with those in which one or both voluntarily restrain from using it for the full 20 trials (see Figure 4). A significantly lower mean joint payoff was earned by those pairs who use the threat bilaterally than by those pairs where one or both subjects avoid the use of threat.

These findings were consistent with expectations derived from the threat-counterthreat es-

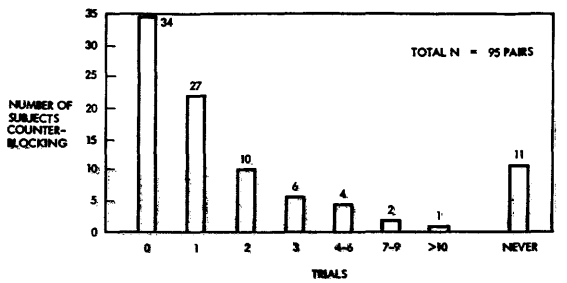


Figure 3. Distribution of Trial Latency of Counter-Blocking.

PAIR BLOCK USAGE	MEAN JOINT PAYOFF	N
BILATERAL	$5.47	114
UNILATERAL AND NEITHER	$6.73	13
M_{diff}	$1.26	

Figure 4. Block Use Related to Joint Payoff for Six Block Conditions.

BLOCK USE FIRST 5 TRIALS	COOPERATIVE AGREEMENT	
	AFTER 5 TRIALS	NEVER
HIGH	18	12
LOW	5	15

N = 50
$x^2 = 5.92$
$(p < .02)$

Figure 6. Frequency of Block Use (First Five Trials) Related to Subsequent Cooperative Agreement.

calation pattern. On the other hand, other findings are clearly at variance with the threat-counterthreat spiraling hypothesis.

1. A comparison of the block and no-block conditions showed the reverse of the predicted threat spiraling. The pairs who had the block action available had a significantly higher joint payoff than did those without it. (See Figure 5.) This result is a direct reversal of the Deutsch and Krauss findings.

2. Again focusing only on the block conditions and those pairs who had not yet reached a cooperative agreement in the first five trials —that is, those pairs who were having some difficulty with each other—we compared those pairs who made *more* use of the block action during these first five trials with those who made *less*. In a complete reversal from expectations, we found that significantly more of the pairs who made extensive early use of block action subsequently reached a cooperative agreement. (See Figure 6.) Nor was there evidence that greater restraint, expressed in delaying the counterthreat response, tended to dampen the escalation of hostility or produce more favorable outcomes over those where less restraint was shown in taking a counterthreat action. We found no significant relationship between counterthreat response latency, measured in number of trials, and joint outcome for each pair. It thus appears that greater restraint in early use or delay in counterthreat response does not dampen the escalation of hostility.

	NO BLOCK	6 BLOCK CONDITIONS
MEAN JOINT PAYOFF	$4.88	$5.60
N	18	108

Figure 5. Mean Joint Payoff for No Block and Block Experimental Conditions.

These results indicated that there is no simple relationship between the use of threats (as measured by blocking) and escalation of hostility (as assessed by difficulty in achieving a cooperative agreement).

Up to this point our data show the following: Subjects who have the block do better than those who do not; yet, those who have the block but do not use it do better than those who use it. This puzzlement is compounded by considering the additional fact just noted. Those who use the block more initially and with less restraint do significantly better than those who use it less or use it more cautiously.

At this point our methodology of computer probes may permit us to do something which was not possible in previous studies. Since the computer has inquired of each subject into his intention and perceptions associated with each use of the block, we may turn to this information instead of engaging in further speculations about the meaning of our results.

3. Generally speaking, how were the blocks intended and perceived? The first striking finding is the frequency with which *first* blocks are used with a cooperative intent—specifically as a signal for coordinative purposes. Approximately 70 per cent of all users included a cooperative reason among the three most important reasons for their initial block usage. In contrast, only 20 per cent failed to register any

cooperative intent, indicating that their use of the block was exclusively intended to threaten or express some reactive hostility.

That these self-ratings did not merely reflect a tendency to characterize one's own intentions in positive terms is indicated by the fact that users with initially cooperative intent do significantly and markedly better in terms of subsequent cooperative agreement than do those who indicate no cooperative intent in the use of the block. (See Figure 7.) Thus, despite the fact that subjects were explicitly given an "individualistic" orientation and sizable sums of real money were at stake, a strongly cooperative orientation was carried into the game by most subjects. These findings suggest that many of the subjects, at least to some degree, substitute cooperative incentives in place of the individualistic ones which the experimenter attempted to induce with instructions and monetary rewards.

Knowing the cooperative use our subjects made of the block action, it is now less surprising that a comparison of the "block" and "no block" conditions showed the reverse of the Deutsch and Krauss results. It is clear, then, that the block action need not always have negative meaning, and it would appear that it can be an actual aid to bargainers if they have the disposition to use it to achieve an optimal coordinate agreement (even, as in the present case, where pre-game ratings showed that our subjects predominantly viewed the block action as a hostile act).

4. Since so many of our subjects achieved an early and apparently easy cooperative agreement—they were willing to share the payoff without conflictual confrontation and needed only to coordinate, not to bargain—we next considered *just* those pairs of bargainers who do not show outward evidence of mutual cooperation (more specifically, those who had not reached a cooperative agreement by the fifth trial) to see what effects threat might have in this fuller bargaining context. These cases should be more interesting and germane for the Deutsch and Krauss hypotheses, for, as a class, these are the bargainers who show evidence of initial interference—whether it be for reason of incoordination or simply conflict of interest in doing so. Among these pairs, one would suppose that those who subsequently reached a cooperative agreement would exhibit an initially lower level of hostility than those who never achieved agreement.

We have already seen that if the frequency of block usage is employed as an *objective* index of hostility, the findings run counter to this expectation; but since we now know that hostility cannot always be ascribed to block usage, let us look directly at the *subjective* indices of hostility for the same groups. We find at this level also there is no evidence for the hypothesis. Pairs which subsequently achieved a cooperative relationship show a *higher* instance of reported hostile intentions and perceptions over the first five trials than those who failed to achieve such a relationship. (See Figure 8.) While not statistically significant, the trend is clearly counter to the spiraling hypothesis.

Finally, let's consider one further possibility. From the threat-counterthreat hypothesis, one should expect escalation in subjective hostility among those pairs who subsequently failed to

RATED INTENTION OF BLOCK	MEDIAN TRIAL OF COOPERATIVE AGREEMENT	N
COOPERATION INDICATED	2.7	69
COOPERATION NOT INDICATED	19.5	24

Figure 7. Intention of First Block Use (Within Pair) Related to Median Trial on Which a Cooperative Agreement Was Achieved.

FREQUENCY OF HOSTILE RATINGS IN FIRST 5 TRIALS	COOPERATIVE AGREEMENT	
	ACHIEVED AFTER FIRST 5 TRIALS	NEVER ACHIEVED
HIGH	16	12
LOW	7	15

N = 50
$x^2 < .10$

Figure 8. Early Subjective Hostility Related to Subsequent Cooperative Agreement.

reach a cooperative agreement and evidence of de-escalation among those who achieved agreement. In fact, increase or decrease in subjective ratings of hostility, both perceived and intended, over the first five trials show no consistent relationship to success or failure in achieving a cooperative agreement (see Figure 9).

All of our findings—the predominantly cooperative intent and perception of block use; the fact that availability of threat is associated with better outcome; the fact that, for pairs that do poorly, neither objective nor subjective indices provide evidence for the escalation of hostility in the early phases of bargaining—all of these would suggest stringent qualifications for the underlying mechanisms of the threat-counterthreat hypothesis if it were to be maintained. If our results run counter to the spiraling hypothesis, what explanation can we offer to account for the varying degrees of difficulty which our subjects had in reaching a bargaining agreement?

It is obvious that pairs of cooperatively oriented bargainers can achieve a jointly acceptable agreement which is profitable to both. It is also the case that competitive pairs (in which both members want to earn more than half of the available payoff) tend to do very poorly.

What happens, however, to those dyads where one member is well-intentioned and conciliatory while the other is out to get as much as he can? A naive extrapolation of the threat spiraling hypothesis would suggest that these pairs should do better than the competitive pairs. We have collected pre-game statements of plans by subjects which provide a basis for such a comparison. We compared competitive dyads with mixed dyads (where a subject planning to share the maximum joint payoff 50/50 is paired with one planning to earn more than one-half the maximum joint payoff). While the competitive pairs were slower to adopt a coordinating strategy, all pairs but one succeeded in doing so within the 20 trials. In contrast, 34 per cent of the mixed pairs failed to do so. Indeed, they have the poorest outcomes of all pair combinations based on pre-game plans. Similar findings were obtained for pair comparison based on personality classification. It is not the dyads which are composed of two players with dominating strategies that have most difficulty; rather, it is those dyads where one member is well-intentioned and conciliatory and the other is dominating or belligerent. (See Figure 10.)

As a tentative explanation of these findings, we propose that the well-intentioned and conciliatory member is initially reluctant to "force" cooperation—to employ his block as threats—and is ambivalent and oscillating when he does so. The subject who delays in responding to a threat with a counterthreat displays to the aggressive member a weak intention to resist and encourages him to persist in his original demands. These then become increasingly unacceptable to the conciliatory member. Two additional findings support this interpretation.

Let's look at the situation of the cooperatively inclined bargainer who initially fails to report seeing any cooperative intent in the other bargainer's use of the block. Should he counter-block immediately or try to nurture coopera-

FREQUENCY OF HOSTILE RATINGS DURING FIRST FIVE TRIALS	COOPERATIVE AGREEMENT	
	ACHIEVED AFTER FIRST 5 TRIALS	NEVER ACHIEVED
INCREASING (TRIALS 4 + 5 > TRIALS 1 + 2)	15	13
DECREASING (TRIALS 4 + 5 ≤ TRIALS 1 + 2)	8	14

N = 50

$X^2 > .10$

Figure 9. Early Increase and Decrease in Level of Subjective Hostility Related to Subsequent Cooperative Agreement.

PAIR COMPOSITION FOR INTERPERSONAL CONCILIATORY FACTOR	MEAN	N
BOTH CONCILIATORY	.52	42
MIXED PAIR	1.86	42
BOTH BELLIGERENT	1.02	42

Figure 10. Mean Number of Joint Zero Profit Trials (Measured over Last Ten Trials for Seven Experimental Conditions).

tion otherwise? Our data show his chances to achieve a cooperative agreement are significantly better if he responds more immediately (within two trials) with a counterblock than if he delays longer in doing so. Twenty-two out of 37 "immediate" counterblockers concur on a cooperative agreement, whereas only one out of seven "delayers" do so.

Finally, our explanation may help to reconcile our otherwise puzzling finding—that greater block usage in the earlier trials is associated with greater likelihood of agreement. On further analysis we find that, on the average, those who never reached agreement used the block less than once every other trial during this period. Thus they employed their block only in a sporadic or intermittent fashion.

In summary, while bargainers who resist the use of threats show the most favorable joint outcomes, these results appear to be related primarily to the pairing of subjects with pregame cooperative dispositions. Rather than spiraling hostilities, a posture of firmness and determination expressed in a consistent and rapid response to threat with counterthreat may contain a belligerent player. Lack of resolution, as expressed in delay and oscillations in employment of counterthreat to an aggressor, may significantly increase the likelihood of escalation of conflict.

In follow-on studies we plan to investigate the conditions under which the exchange of threat and counterthreat actions lead to conflict escalation and/or to a dampening or reversal of the conflict; to identify "points of no return" if these are present; to assess the effect of massive retaliatory capability on the use of smaller threat actions; to determine under what conditions such a capability would exert a stabilizing or destabilizing influence on negotiations.

In pursuing these goals, we also plan to exploit further the unique advantages afforded by our computer approach. We are planning improvements in two major directions: first, to improve techniques for eliciting subjective data through computer probes so as to minimize their interrupting features and increase the value of the information elicited; second, to gain experimental control through computer simulation of one of the players in each pair. This will permit direct investigation of effects of individual styles of play (ruthless competitor, strategic cooperator, irresolute cooperator, moral pacifist, etc.); this, in turn, will provide greater control of intermediate critical stages of the game, thus reducing the obscuring effects of unique patterns of play and improving ability to evaluate and compare results for different bargaining pairs.

More important even than the specific findings in this study is the demonstration of the potential usefulness of the computer as a tool for gathering critical data in studies of social interaction. As a result, data become available which have been virtually unobtainable on a scale necessary for scientific investigation.

REFERENCES

1. DEUTSCH, M. Trust, trustworthiness and the F scale. *J. abnorm. soc. Psychol.*, 1960, *61*(1), 138-140.
2. DEUTSCH, M., and KRAUSS, R. M. The effects of threat upon interpersonal bargaining. *J. abnorm. soc. Psychol.*, 1960, *61*(2), 181-189.
3. DEUTSCH, M., and KRAUSS, R. M. Studies of interpersonal bargaining. *J. conflict Resolut.*, 1962, *6*, 52-76.

REAL TIME QUICK-LOOK ANALYSIS FOR THE OGO SATELLITES

R. J. Coyle and J. K. Stewart
Datatrol Corporation
Silver Spring, Maryland

Introduction

The National Aeronautics and Space Administration has designed a series of general purpose orbiting satellites which have the family name of Orbiting Geophysical Observatory (OGO). Each of these satellites is capable of carrying up to 50 scientific experiment devices, which transmit data to ground stations via a common telemetry channel. There are two tracking stations, at Rosman, North Carolina, and Fairbanks, Alaska, which will receive this telemetry, transmitting it to Central Control at Goddard Space Flight Center in Greenbelt, Maryland, at a data rate of up to 64 KC.

General Description of the Programming System

The requirement of the programming system for OGO was to provide quick-look analysis and control of the status of the spacecraft and selective experiments on board the satellite. (The tracking and orbit determination of OGO is not a function of this program.) All telemetry is recorded for further extensive analysis in non-real time on other equipment. There are several special purpose consoles attached to the computer to accept telemetry, and provide a means of communication to and from the spacecraft. These are appropriately called: PCM Telemetry input equipment, Control and Display Console, and Command Console. It is through the integration of these consoles with the computer program that experiments on board the satellite may be selected to start, or terminate, by a command from the computer. The selective analysis of those experiments transmitting data is initiated by an input request from the Control and Display Console and selective spacecraft status analysis is handled in the same way. It is therefore a requirement of the computer program that it be flexible enough to handle these many and various requests instantaneously. Since the bit rate of 64 KC means a new telemetry frame of data will arrive every 18 ms., the computer programs must be able to make maximum use of the capabilities of the computer.

The SDS 920 computer was chosen by NASA to handle the quick-look analysis and control for OGO. The 920 is a small-scale computer with a 24-bit word and an 8 us. cycle time. It has one index register and uses single-address fixed-point arithmetic logic. It is capable of handling many kinds of input/output, to include: magnetic tape, paper tape, punched cards, on-line typewriter, and high speed printer. The OGO 920 has 8 K of memory and 32 channels of interrupts for input/output use. The OGO installation is illustrated in Figure 1.

Functions of the Real Time Monitor Control System

The programming system for OGO can be thought of in two distinct parts: the Real Time Monitor Control, and Experiment Processors. All routines necessary to connect the computer with the external environment are integrated into the Real Time Monitor Control system. It

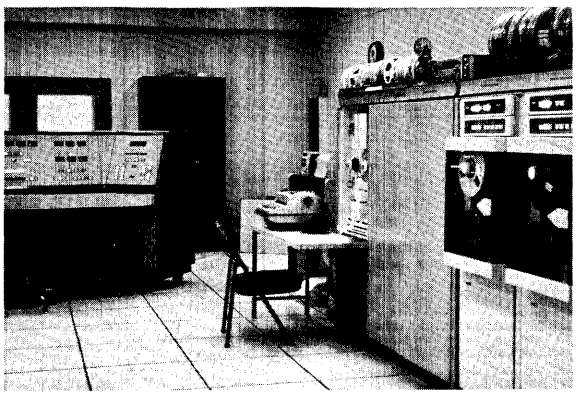


Figure 1. OGO Central Control Installation.

must service all interrupts, provide for the receipt of all input including telemetry and manual input requests from the consoles, direct and send out all output to the appropriate receiving device, keep track of time, and determine the sequence in which all functions and processing is conducted. The Monitor Control does not process any of the data itself, but rather acts as a general purpose framework in which selective routines perform the processing. These selective routines are called Experiment Processors and are independent routines operating under control of the Monitor. These routines process telemetry input data, returning to Monitor formatted output for driving digital displays on the special purpose consoles, for printing, and for typing, as well as other information to the Monitor.

The first distinct part of the programming system for OGO, the Monitor Control, is itself divided into three sections: the Schedule Program, Monitor Processors, and Interrupt Processors. The following is a definition of the functions and characteristics of these various sections.

Monitor Schedule Program

The Monitor Schedule Program consists of routines which collectively coordinate, supervise, and schedule all processors in the system, utilizing a priority table of processors. It saves the status of the machine whenever an interrupt occurs, and facilitates a proper return from an interrupt by restoring the condition of the machine at the point of interrupt. The heart of this scheme is the Monitor Schedule Routine whose function is to examine sequentially the entries in the priority table in order to determine the next routine of highest priority to be processed.

The priority table of processors consists of a group of words, or module, which identify and describe each processor with a reference in the table. Due to the difference of functions of the various processors, the modular approach allows for flexibility in assigning priority during the debugging stage of the system. By chaining these modules together, say by the first word of each module referencing the first word of the next module, the modules may be of various length (Fig. 2).

In order for the Schedule Routine to carry out its function of determining the next processor of highest priority with something to do, it must know the status of each processor in the

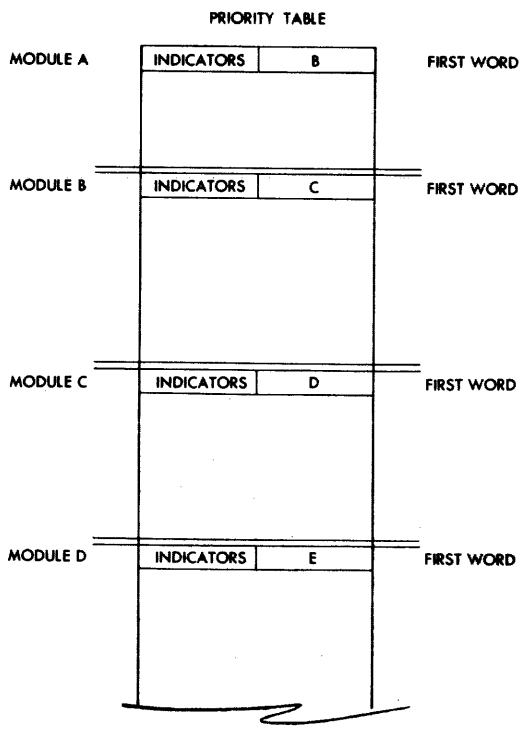


Figure 2. Make Up of First Word of the Modules.

priority table at any given instant of time. This is accomplished through the use of indicator bits in the module, say the left portion of the first word of each module (Fig. 2). The various states that a processor may be in are the following:

'in process' —This processor was interrupted while it was processing, and entry should be made at the point of interrupt.

'ready' —this processor now has something to do; entry should be made at the beginning.

'suppressed' —this processor should not be processed at this time.

'terminated' —this processor has now completed its processing.

'not in core'—this module is available for the addition of a new processor.

These indicator bits, then, inform the Schedule Routine whether or not the referenced processor is now available for processing. Since the priority table entries must be examined from the top each time return is made to the Schedule Routine, a rapid method of examining the indicator bits of each module was needed. To clarify the method that was devised, it is necessary to diverge for a moment and describe two uses of an instruction in the SDS 920 repertoire, which may be used to load the index register.

```
      EAX    A
      X Reg. = A
```

Immediate Addressing, where the address field of the instruction itself is placed in the index register.

```
      EAX*   A
A     PZE    1
      X Reg. = 1
```

Direct Addressing, where the contents of location A, referenced in the address field of the instruction, is placed in the index register.

```
      EAX*   A
A     PZE*   B
B     PZE    2
      X Reg. = 2
```

Indirect Addressing, where, if the contents of location A is in turn indirectly addressed, the address field of A will again be used as the location from which to load the index register.

The SDS 920 computer has the capability of unlimited indirect addressing; that is, it is possible to continue the above indirect addressing of an instruction for many steps, until a location is encountered that is not itself indirectly addressed. It is convenient then to choose the indicator bits of the modules discreetly to take advantage of this feature of the machine as well as the fact that the modules are chained together. If all states of a processor which indicate that it not now available for processing contain the indirect addressing bit as one of the indicator bits, then only those routines with something to do at the current time would be picked up for examination. For instance, the processor states of 'suppression,' 'termination,' and 'not in core' all result in there being nothing to do at this time on this particular processor.

In Figure 3, the consequences of giving an "EAX* A" would be that modules A, B, and C would be skipped and the first word of module D would be placed in the index register. For module A is 'suppressed,' module B is 'not in core' and module C is 'terminated," indicating that no processing should be done on these processors at this time. The contents of the index register actually contains the location of

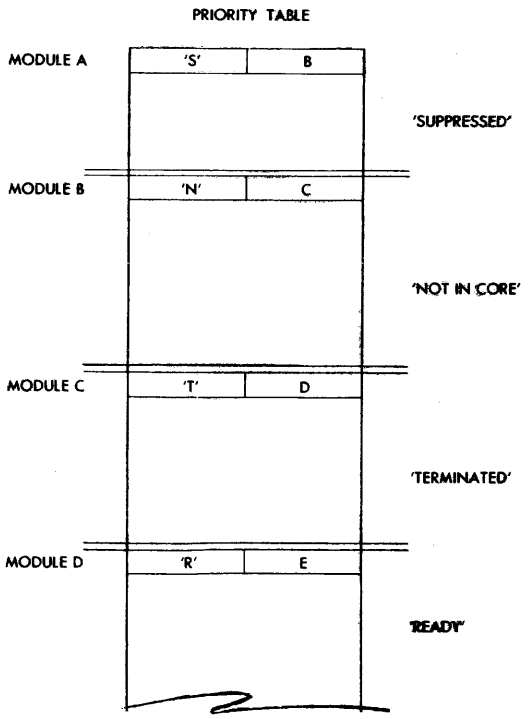


Figure 3. Example of the Use of the Indicators.

the next module, due to the chaining criteria, but this presents no particular problem if reference is always made to the words of the module in reverse; that is, -1,2 or -6,2 etc.

Since it costs only one machine cycle to skip each successive module whose indirect addressing bit is set, the result is a most efficient search of the priority table. The alternative would be to pick up the contents of the indicators of each module and then through various logical compare instructions determine whether any of these situations are the case. It is apparent that the scheme using the indirect addressing feature of the computer facilitates spending the minimum time examining the entries of the rather lengthy priority table of processors.

In addition to the indicator and chain word, each module also contains the first word location of the processor which it references, as well as a block of locations in which the condition of the machine is saved when this processor is interrupted. If the referenced processor has input/output functions, the module will additionally contain the location of the I/O data which is stacked in a table for subsequent transfer or editing. If the processor is an Experiment Processor, it will also contain the location of flag words in the processor by which communication is made with the Monitor. That is, words by which each Experiment Processor can signal Monitor of its various output requirements, and whether it has completed processing, etc. (Fig. 4).

The many processors of the Real Time Control System may be in various states during the course of real time operations so that it is impossible for one processor to communicate directly with another. This is especially true of I/O data and channel select requests. Therefore, two subroutines were devised in the Monitor Schedule Program to handle the passing of data from one processor to another. One subroutine handles the stacking of data onto the receiving processor's stacking buffer, while the other handles the unstacking of the next data to be processed. The barrel, or wraparound, method was chosen to facilitate the stacking and unstacking from these buffers. That is, a 'first in-first out' list type of scheme. To avoid moving the data around in core memory, since

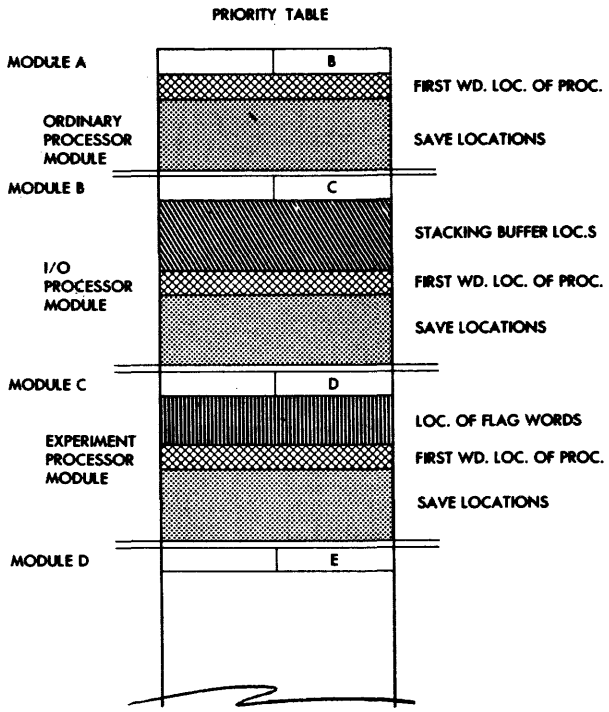


Figure 4. Composition of the Modules.

this is necessarily inefficient, the items which are stacked and unstacked in the buffers are the locations of the I/O data or channel select requests.

Each I/O processor's module, then, contains the top and bottom locations of the stacking buffer for that processor. It can be seen that these buffers may be variable in length from one processor to the next since each processor's buffer is uniquely defined in its module. The module also contains the location of the first item to be unstacked as well as the next free location into which to stack. These locations may vary anywhere within the prescribed stacking buffers area, hence the name 'barrel method' (Fig. 5).

When an item is unstacked, the 'first' location reference in the module is replaced by the 'first + 1' location reference so that the next item is now referenced for unstacking. Similarly, when an item is stacked in a buffer at the 'next' location, the 'next' location reference is then updated to the 'next + 1' location as the next free location in which to stack. The top and bottom locations, delineating the total

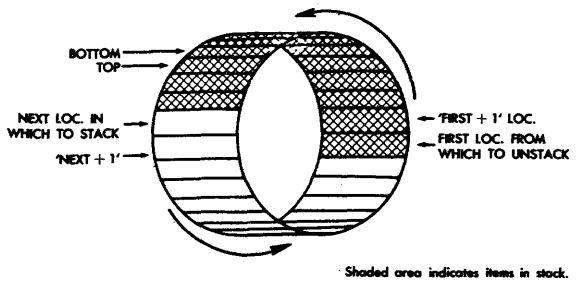


Figure 5. 'Barrel-Like' Composition of Stacking Buffers.

buffer area, can be thought of as being only one unit apart, for the subroutines always treat the bottom location + 1 as if it were the top location. The circular action has now been effected.

Monitor Processors

The Monitor Processors handle the input/output requirements of the Real Time Monitor Control, as well as a few other minor functions. The SDS 920 has two buffered channels for input and output data transfer and 32 real time channels, all of which have interrupt capabilities. One of the buffered channels is used exclusively for the telemetry input, the other handles all other I/O functions, except those from the special purpose consoles, time interrupts, etc. Due to the difference in the rates of speed of the devices attached to this I/O channel, it was decided to have a series of processors, each one associated with only one I/O device, rather than a single large I/O processor geared to the buffered channel. Each of these processors, like all other processors, is under control of the Monitor Schedule Program. Its priority order is determined by the position of its module in the priority table and is in direct relationship with the speed of the I/O device to which it refers. For example, the priority of a processor handling the on-line printing is higher than the one handling the output messages to the typewriter.

Since only one channel select request can be executed by a buffered channel at a time, all of the Monitor I/O Processors hand off their I/O select requests to a single processor, rather than randomly selecting the channel themselves. The function of this processor is to select the channel with whatever request is next in line, regardless of the kind of I/O device indicated. This processor is guaranteed that the channel is ready to receive the select request, for as each request is serviced, this processor 'suppresses' itself. That is, it sets the indicator bits in its own module so that it will not be considered for processing again until it is 'unsuppressed.' When the interrupt from the completion of the data transfer on that channel occurs, the interrupt processor servicing this interrupt will 'unsuppress' the channel selecting processor so that it may now continue with the next I/O request, since the channel is free to be reselected. This channel selecting processor has the highest priority of all the other Monitor Processors, thus allowing for the maximum use of the channel at all times.

There are other Monitor Processors which handle the input/output to and from the special purpose Control and Display Console and the Command Console. These processors are controlled and executed in a straightforward manner since each function is handled by a unique interrupt through the 32 real time channels. There are eight such processors connected with these consoles. One additional Monitor Processor handles time in the system. It maintains a gross time figure in minutes, and prints out a GMT time message periodically. It is especially used to flag certain processors dependent on some interval of time that it is time to commence processing. This procedure is called 'readying' a processor.

Interrupt Processors

The third category of the Monitor Control deals with processors which handle interrupts. Each Interrupt Processor is coordinated with a separate and unique interrupt channel. The 32 channels of interrupts have a priority system of their own in that lower level channels may be interrupted by any higher level channel. This means that the corresponding Interrupt Processors also may be interrupted during the course of their processing if an interrupt of a higher priority comes in. Therefore, the Interrupt Processors are naturally coordinated with the demands of the interrupt to be processed as well as with the priority level of the channel selected to handle that interrupt. For instance, it was decided that the telemetry interrupt would be on the channel with the highest priority, since a new word of telemetry comes into

the computer every 280 us. The one second time interrupts were given the second highest priority, and so on.

Since all processors, whether Monitor Processors, Interrupt Processors, or Experiment Processors, return control upon completion to the Monitor Schedule Routine, it was necessary to vary this requirement somewhat in the case of Interrupt Processors. For if an Interrupt Processor is itself interrupted, then return should be made to the point of interrupt. For interrupts are a demand to acknowledge some I/O data transfer and must be processed before the normal processing continues. This allows for smooth and coordinated interaction between the different parts of the Real Time Monitor Control system. The technique of drawing a hypothetical line in memory was used to allow an Interrupt Processor to decide quickly whether it had interrupted out of another Interrupt Processor, for all Interrupt Processors are placed above this 'line.' If this is the case, return is made to the interrupted Interrupt Processor and not to the Schedule Routine. It logically follows that when all Interrupt Processors have finished, the one which caused the first interrupt in time will be the last to finish and will return control to the Schedule Routine. One of the advantages of this scheme is that it is not necessary for the Interrupt Processors to run in a 'disabled' mode. That is, where the computer is prevented from receiving another interrupt by executing a special instruction. In fact, the 'disable' instruction is given rarely in this real time system, and when it is used, the duration of the disabled mode is very short. For, since a new telemetry word comes into the computer every 280 us., it must not be disabled for a period of time equal to or greater than that.

Experiment Processors

The second distinct part of the programming system for OGO consists of all those selectable, independent routines called, collectively, Experiment Processors. The function of these Experiment Processors is to perform analysis on the telemetry input data, both of the experiments on board the satellite and the status of the spacecraft itself. These processors operate under control of the Monitor Control programs, returning, upon completion of processing each telemetry frame of data, to the Monitor Schedule Routine. Since there are many more Experiment Processors than would fit into core memory along with the permanent Monitor Control, it is the function of the Monitor to be able to add and delete selective Experiment Processors without disturbing the real time processing. A Monitor Processor, called the Real Time Load Processor, performs this function. It is this real time load capability of the Monitor Control which affords the flexibility of operation required of the system. The Monitor Control can thus provide for maximum use of memory capacity as well as maximum use of processing time.

Requirements of Experiment Processors

All Experiment Processors place three general requirements on the Monitor. They must be able to obtain all data needed from the Monitor, including such parameters as time and orbital position of the satellite, as well as specific data points from the telemetry frame. They must also be able to return the results of their processing back to the Monitor in the form of messages to be output on the on-line printer. Finally, they must be able to communicate with the Monitor that they have completed their processing or to request that special functions be accomplished, such as sending commands to the special purpose consoles. It is the function of the Real Time Load Processor to provide the capability of bringing such an Experiment Processor into memory when requested by depressing the appropriate button on the Control and Display Console, and to make all necessary connections and links between the processor and Monitor.

Coding Rules

A problem arose in the fact that these Experiment Processors were to be written by many different programmers. This would appear to involve a considerable amount of supervision of the coding used to protect the system from 'blow ups,' but actually it was possible to handle this checking automatically by writing a service program called the Static Checker. As its name implies, the Static Checker checks the symbolic coding of an Experiment Processor to determine whether the programmer has complied with the various coding rules laid down for the system.

The rules to be followed by a programmer when writing an Experiment Processor are of two types: real time restrictions, which are necessary to protect the Monitor Control; and relocatable binary restrictions (since the processor will be used in this form), which follow from the nature of relocation itself. The rules are:

Real Time Restrictions

1. No I/O instructions
2. No breakpoint switch tests
3. No halts

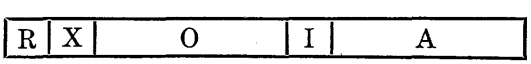
Relocatable Binary Restrictions

1. No addresses of type: —Symbolic, Symbolic + Symbolic, or use of * or /
2. No references to absolute memory locations
3. No negative constants or masks

The real time restrictions are quite straightforward, since all I/O is handled by the Monitor; breakpoint switches (sense switches) are reserved for Monitor use, and halts are strictly anathema in real time work.

The first set of relocatable binary restrictions are the familiar ones inherent in relocation and the second restriction, forbidding absolute addresses, is also fairly standard in this kind of work. The third restriction, however, is somewhat unusual and arises from the unique manner in which relocation is handled in the SDS 920. A brief look at the makeup of an SDS 920 instruction word will serve to illustrate this.

A 24-bit instruction word in the 920 is broken down as follows:



The fields are, from right to left, a 14-bit address field, one bit for indirect addressing, a 7-bit operation field, one bit for indexing, and finally, the left-most bit which is not interpreted when the instruction is executed. That is, the instructions 0 35 01000 and 4 35 01000 are interpreted by the computer as being identical. Since the address field is the only one that need be relocated in any instruction, the left-most, or sign bit of the word may be used for

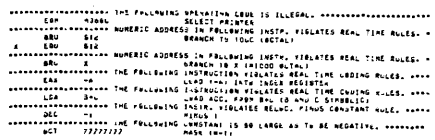


Figure 6. Example of Static Checker Diagnostics.

this purpose, and this is, in fact, the scheme commonly used on the 920.

For data words, however, the full 24 bits are used and it is for this reason that Experiment Processors needing negative constants or masks with a left-most one must generate them. This is the only real coding restriction placed on the programmer and in practice is not serious, since the 920 has the capability of forming both 1's and 2's complements with single instructions.

The Static Checker accepts a symbolic deck and produces a listing containing error diagnostics, if any (Fig. 6).

Formatting

It was apparent that the most straightforward way to maintain the Experiment Processors for use with the real time system was in the form of a library on magnetic tape. In line with this, two fundamental decisions were made concerning the physical makeup of these processors, both related to the efficient use of the computer's memory at execution time: they were to be in relocatable binary (as previously mentioned) and they were to be broken up into uniform-length blocks. Since it was considered unreasonable to require programmers to code their Experiment Processors in this latter manner, it was necessary to devise a method to take the programmer's processor, coded in a single sequence, and convert it to the desired form automatically.

In addition to checking the Experiment Processor, this conversion to uniform-length blocks is also provided by the Static Checker during the same run. This was easily facilitated, since the Experiment Processor is input to the Static Checker in symbolic form and the necessary additional instructions and pseudo-

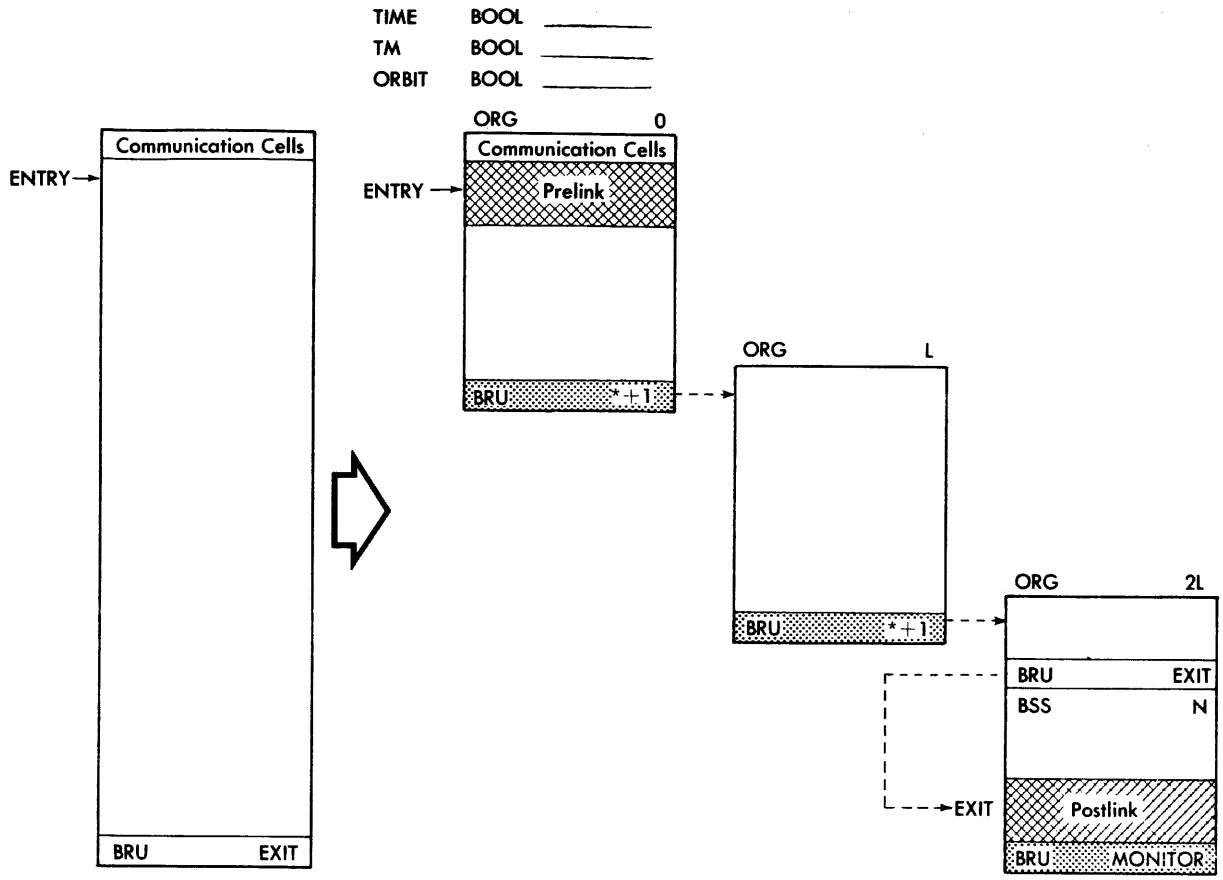


Figure 7. Conversion to Uniform-Length Blocks by Static Checker.

operations may be inserted in that form (Fig. 7).

It will be noted that a few extra things have been inserted as well. For instance, the Experiment Processor obtains telemetry and other data from the Monitor by referencing cells which are absolutely located in the Monitor. Therefore, these locations are defined by BOOL pseudo-operations inserted by the Static Checker; the programmer need only refer to them symbolically. The Prelink and Postlink, which are bookkeeping linkages to the Monitor, are also inserted at this time.

The breaking up of the consecutive sequence into uniform-length blocks is effected by inserting an ORG (origin definition) pseudo-operation at fixed intervals. The actual connection between one block and the next is handled by a 'dynamic link,' the BRU (unconditional branch) instruction which is inserted at the end of each block. The advantage to be gained by this approach is that the blocks of a processor need not be contiguous in memory, and in practice this is usually the case. ORG's are used to define the start of each block in order to ensure that the result of assembling the symbolic output of the Static Checker is a series of paper tape records of uniform length, simplifying the maintenance of the library of Experiment Processors on magnetic tape.

The problem of pseudo-operations which define more than one memory location, such as:

DEC	1,2,3	Places decimal 1-3 in 3 successive locations
OCT	4,5,6,7,10	Places octal 4-10 in 5 successive locations
BCI	4,ALPHA FIELD	Places BCD characters in 4 successive locations
BSS	20	Reserves 20 successive locations.

is handled by a special diagnostic in the Static Checker which flags any such pseudo-operation which causes the 'straddling' of two blocks as a coding error. The programmer must then rearrange his processor so that the locations defined will fit into one block.

Other Service Programs

The Static Checker has a companion program, the Dynamic Checker, which works with an Experiment Processor in Machine Language: the result of assembling the output of the Static Checker. The Dynamic Checker is essentially a simulator which provides the opportunity to debug an Experiment Processor under conditions as similar as possible to those of the real time Monitor. The Experiment Processor is embedded in the Dynamic Checker and is 'fed' telemetry frames via the same communication cells used in the real time case. The output of the processor is printed for subsequent analysis by the programmer.

The Dynamic Checker uses for input a magnetic tape containing simulated telemetry frames. Its modus operandi is to read a frame, branch to the Experiment Processor, and check to processor's communication cells for commands and/or output, repeating this cycle until the tape end-of-file is reached. The input tape is generated by another service program, the Tape Builder, which contains a pseudo-random number generator and which, essentially, generates a magnetic tape of simulated telemetry frames containing nothing but white noise. This is to check the Experiment Processors for one of their basic requirements: that they be capable of accepting any data and still not 'blow up.' The Tape Builder, however, has the provision of accepting programmer-coded subroutines which allows the data generated by these subroutines to be inserted in specified words in the telemetry frames. In this manner, the programmer may insert realistic data in the words of the telemetry frame which are analyzed by his Experiment Processor and thus may use the Dynamic Checker as a diagnostic tool.

Like the Static Checker, the Dynamic Checker has a second function in addition to a debugging one. This is the function of timing the Experiment Processor. One of the two interval timers available on the SDS 920 is used for this purpose; the result being that, at the end of the run, the Dynamic Checker prints out the worst-case (longest) execution time in machine cycles for the Experiment Processor being tested. As will be seen later, the two parameters: number of blocks (given by the Static Checker) and worst-case execution time (given by the Dynamic Checker) are the ones which will determine whether this particular processor may be loaded and executed at any particular time when the real time system is running.

In addition to the above service programs, there is, of course, a Tape Librarian program whose task is to maintain the library tape of Experiment Processors; adding, deleting, or replacing processors as appropriate. In addition, the Tape Librarian includes a computed checksum with each block written on the library tape. These service programs taken together provide a means of carrying an Experiment Processor from the coding stage through inclusion on the library tape in a fairly automatic manner (Fig. 8).

Loading Experiment Processors in Real Time

The Monitor has two restraints which determine whether an Experiment Processor may be

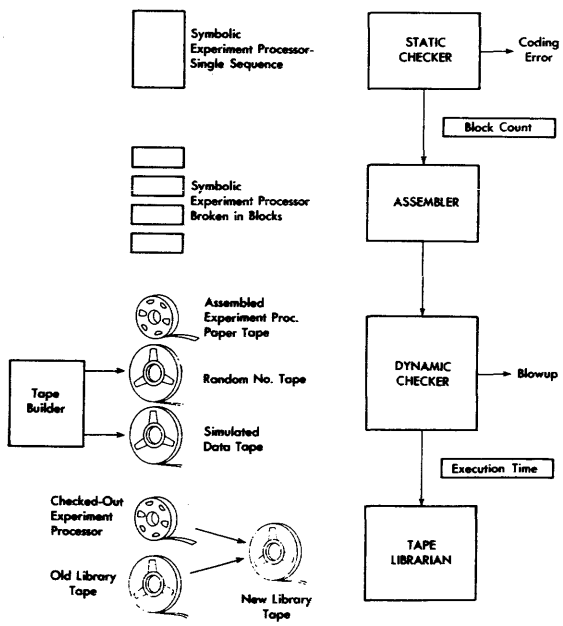


Figure 8. Flow of Experiment Processor from Coding to Library Tape.

added at any given time: whether there is room in memory and whether there is enough time to execute it. The portion of memory available for use by Experiment Processors, is called free storage and is allocated in blocks of the same size as the subdivisions of these processors. The amount of time available for Experiment Processor execution is called slack time and is the number of machine cycles presently not being used in the 18 ms. interval between successive frames of telemetry. A parameter table is maintained for use by the Real Time Load Processor containing one word for each processor on the library tape. Each parameter word contains two fields which specify the amount of memory (in blocks) and amount of time (in machine cycles) required by that particular Experiment Processor. When a request is entered into the computer to add a particular Experiment Processor, this request is handed off to the Real Time Load Processor which may easily check whether there is enough free storage and slack time to accommodate this Experiment Processor. If not, the request is placed in a special stack called the Request Stack. If the processor can be added, however, the Load Processor positions the library tape, reads the Experiment Processor into memory, relocates it, and connects it to the Monitor. Actually, the Load Processor makes requests to the Monitor for every tape operation given and it is the interrupt that occurs after each record is skipped or read which causes the Load Processor to be entered again. (This includes backspacing and rereading in case of a checksum error.)

If the request given to the Load Processor is to delete an Experiment Processor, either because this request has been made from the console or because the processor has signalled the Monitor that it has finished its run, then the Load Processor locates this processor, disengages it from the Monitor, returns the processor's blocks to free storage, and adds the processor's execution time to the slack time count. The Load Processor then checks its Request Stack to see if there are any requests to add processors which now may be accommodated.

The above description is considerably abbreviated and does not cover some of the special considerations of the Load Processor as actually written, such as error checking, backspacing, and rereading due to checksum error, etc. Also, although it is convenient to speak of 'picking up' blocks of free storage and 'returning' them to free storage, nothing is moved around in memory the change is actually one of allocations.

The basic pieces of information the Load Processor works with are four in number (Fig. 9). PARAM is a parameter table giving the storage and time requirement for every Experiment Processor on the library tape. The block count is in the operation field and the execution time is in the address field. Thus, as illustrated, Experiment Processor 34 requires 3 blocks of storage and has a worst-case execution time of 700_8 cycles.

Figure 9 also shows SLACK, which is a cell containing the available slack time in cycles, 1300_8 here. FREE is a cell which refers to Free Storage, the fourth element. It can be seen that an elementary list processing technique has been employed here. The address of the first

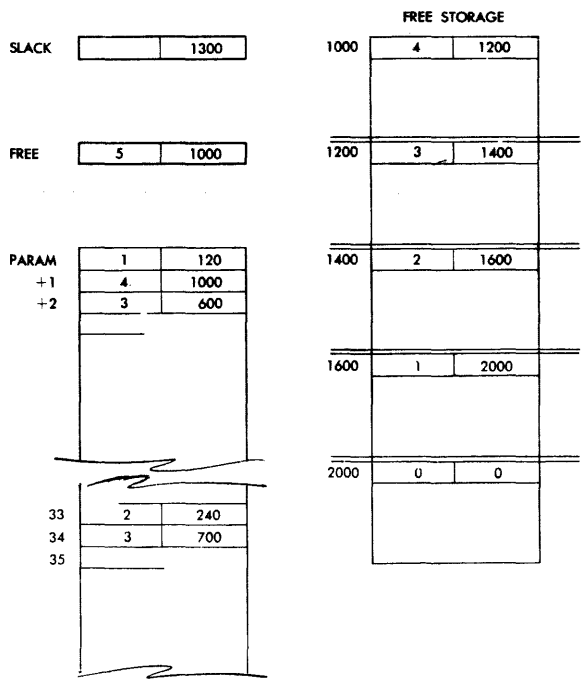


Figure 9. Elements Used by the Monitor Load Processor.

word in each block in free storage contains the starting location of the next so that they are chained together with the address of FREE giving the starting location of the end block in the chain. Furthermore, from bottom to top in the chain, the operation field of the first word in each block contains a serial count, starting at zero. This count is also carried over to the operation field of FREE and thus this cell not only provides a link to all of the blocks in free storage but also contains the number of blocks in the chain. The linking of blocks shown is somewhat idealized. In actual practice, these links become quite 'scrambled.'

Under this scheme, the process of testing whether a requested Experiment Processor may be added is handled quite easily, simply by using its parameter word from the table PARAM: the address field of the parameter word (execution time) must be less than or equal to the address field of SLACK, and the operation field of the parameter word (block count) must be less than or equal to the operation field of FREE.

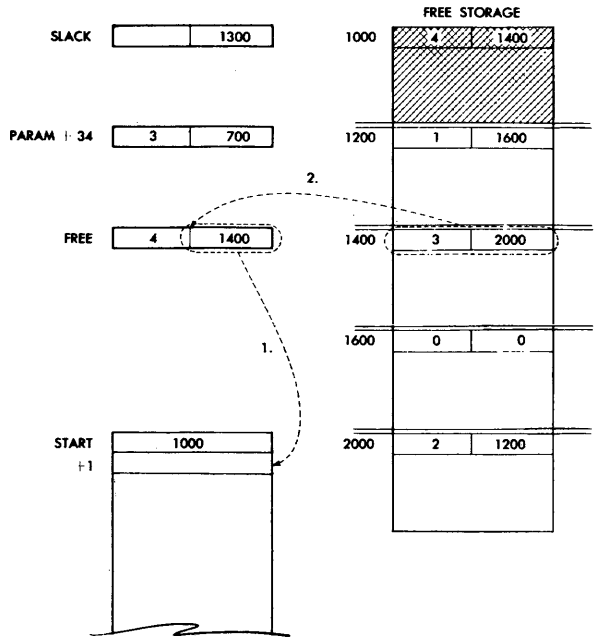


Figure 10b. 'Removing' Blocks from Free Storage—Second Step.

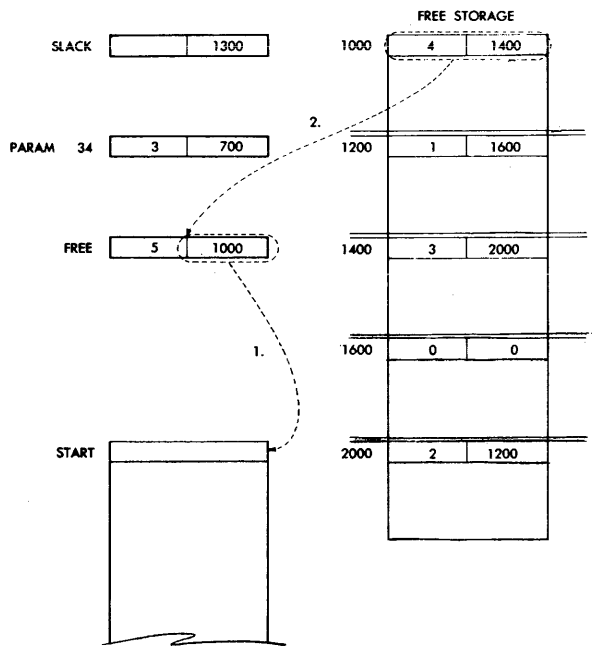


Figure 10a. 'Removing' Blocks from Free Storage—First Step.

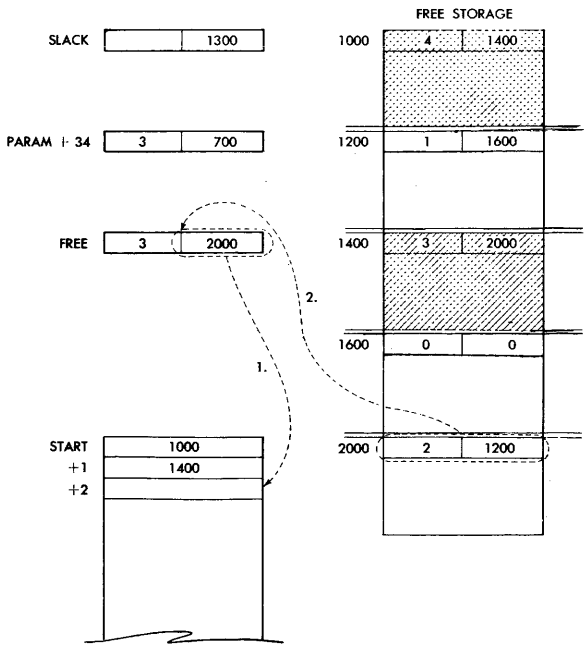


Figure 10c. 'Removing' Blocks from Free Storage—Third Step.

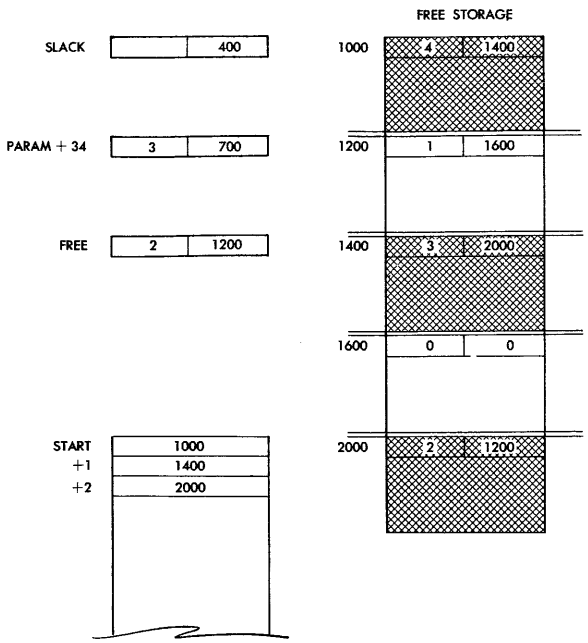


Figure 10d. 'Removing' Blocks from Free Storage—Final Condition.

Adding an Experiment Processor

In order to add an Experiment Processor, blocks must be 'removed' from free storage so that the processor may be read into them (Fig. 10). The 'removal' of a block is a two-step process, in which the address of FREE is stacked in a table called START and the contents of the location given by that address (the starting location of the block) is then stored in FREE. FREE now refers to the next block in the chain, and this process may be continued as often as necessary. This is, of course, the familiar 'popping up' of list processing and here it is done three times since Processor 34's parameter word calls for three blocks. The final step is to subtract the execution time in the processor's parameter word from SLACK, setting it to the unused time now available.

The table START is used for two things. First it is used to specify the blocks of memory into which the Experiment Processor is to be read. START is also used for relocation with a companion table BITS which is generated to contain successive multiples of the block length, starting at 0. Since the Experiment Processors are assembled with a base origin of zero, it can

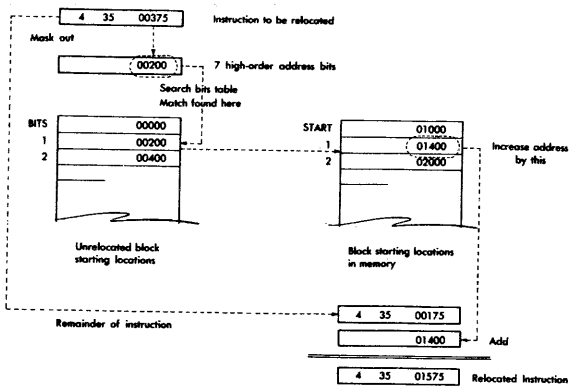


Figure 11. Example of Relocation.

be seen that BITS consists of the unrelocated starting location of each block. Figure 11 shows how this relocation is carried out on a sample instruction.

The method of relocation allows an instruction in any block to reference a location in any other block occupied by that processor. The dynamic link between blocks could then be effected by the final BRU *+1 in each block, simply by flagging these instructions for relocation. In practice, however, the Tape Librarian program replaces these terminal branches with computed checksums when an Experiment Processor is added to the tape library. The Load Processor uses these checksums to determine whether the Experiment Processor has been loaded properly, with the usual re-reads in case of failure. When the processor has been successfully read in, the relocation sequence in addition replaces the checksums with the original branches, connecting each block to its successor and the last block to the Monitor.

It might be pointed out that there is no fixed requirement that the blocks have a length 200_8 nor, in fact, need their length be a power of two, although this simplifies the relocation scheme. This figure was chosen as a convenient length and may be changed if experience with the system warrants it.

Deleting an Experiment Processor

Figure 12 shows how the blocks of the Experiment Processor added in the example above are 'returned' to free storage. The starting location of the processor is obtained from the

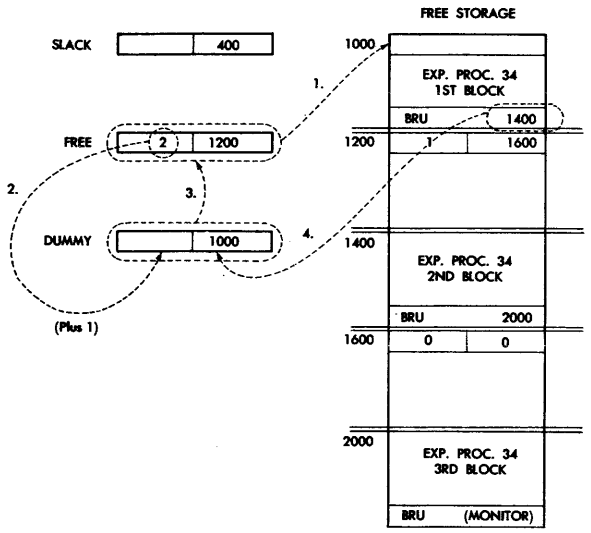


Figure 12a. 'Returning' Blocks to Free Storage—First Step.

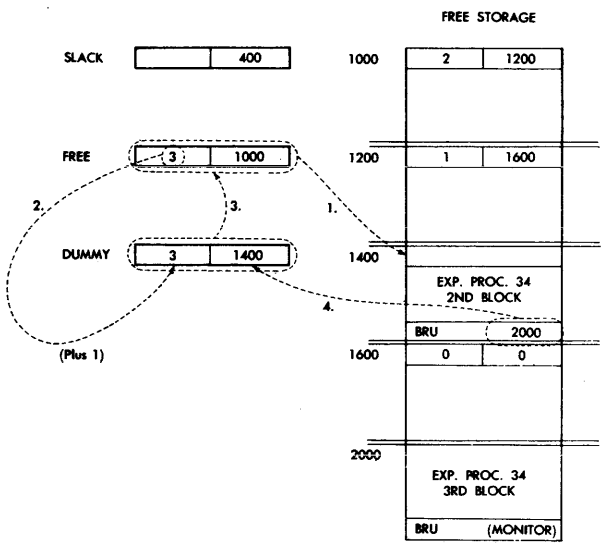


Figure 12b. 'Returning' Blocks to Free Storage—Second Step.

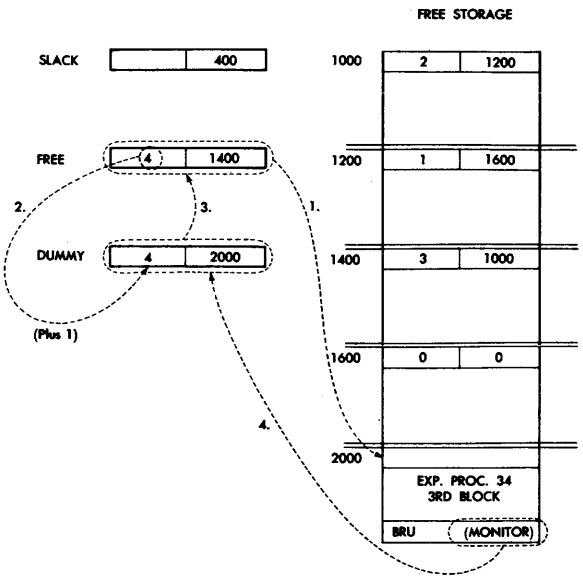


Figure 12c. 'Returning' Blocks to Free Storage—Third Step.

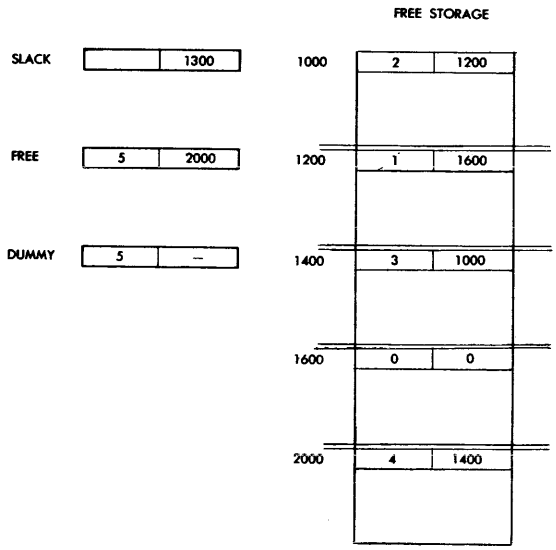


Figure 12d. 'Returning' Blocks to Free Storage—Final Condition.

module in the priority table to which it was assigned and placed in a working cell, called DUMMY here. Since the terminal branch at the end of every block provides a link to the next block, this 'pushing down' process can be carried out in a four-step process analogous to the one in which blocks were 'removed' from free storage. The block count is updated in the process so that at completion FREE again contains the correct block count when the number of blocks given by the processor's parameter word are 'returned' to free storage. Finally, the processor's execution time is added to SLACK to set it to the unused free time now available.

As shown by these examples, program blocks are never actually moved in memory; they are

simply read into the first available blocks and, since the blocks are dynamically chained with branch instructions, these blocks need not be contiguous in memory. A processor is deleted essentially by disconnecting it from the Monitor and allowing the storage it occupies to be reused. Since the Tape Librarian program converts all block reservations in an Experiment Processor to the corresponding number of zero words when the processor is added to the library tape, these memory blocks need never be cleared.

Conclusion

We feel this project is of interest since it represents an unusual situation in data processing. This is due to the fact that, while there was a requirement to selectively process different data points on demand, there was not a need to process all of the data available. Given the selection of the specific computer, however, this project attempted to optimize its use by providing the capability of doing the maximum useful work that memory space and processing time will allow. In this paper we have pointed out techniques to minimize the execution time of a powerful Real Time Monitor and to allocate reusable storage in a flexible and efficient manner. While using the computer to sample and monitor data is not a typical data processing application, it is hoped that this paper has indicated the efficiencies which we feel are inherent in the real time approach.

AN ETHOS FOR THE AGE OF CYBERCULTURE*

Alice Mary Hilton
President
A. M. Hilton Associates
New York, N. Y.

PART ONE—NEED FOR A NEW ETHOS

I. *Introduction*

"Acceptability," said John Kenneth Galbraith, "is the hallmark of the conventional wisdom."[1]

Since, however, the pronouncement of the conventional wisdom is the prerogative of those in eminent public, academic, business or labor positions, I am not privileged to bore you with a recital of the conventional wisdom, and—even if I could do so—to entertain you by expounding it at a properly sophisticated level. I must, therefore, look to that arch enemy of the conventional wisdom—the march of events.

If ever a period in the history of man demanded radical—I am using 'radical,' derived from *radix,* root, in its original sense, namely, going to the roots—fundamental wisdom, it is surely this revolutionary period of transition to a new era—the age of cyberculture—the new era that is formed by a science, cybernetics, born barely a quarter of a century ago, and a technology that, for all its precocious development, has barely left the cradle.

Most of us in this room were probably proud midwives assisting in the delivery of the computing machine only a decade or so ago.

Since then, the world has changed radically. Three powerful new phenomena have precociously reached their vigorous, boisterous adolescence—long before the world is prepared for the scientific-social-technological-economic-cultural revolution that has been unleashed. Those in the center of any revolution are always the least disturbed. The hub of a wheel is fairly stable, the eye of a hurricance is calm, and those who create the concepts and forge the tools of complex social revolutions are neither alarmed by the enormous power of their brain-children, nor are they surprised.

Never has a powerful and complete revolution developed more quickly than this cybercultural revolution that is affecting the lives of millions of human beings who have never even heard the new words to describe powerful new concepts. In fact, things have been happening so fast that even those who know a great deal about one of the phenomena have not had time to learn enough about the others—or about the world they are changing.

First among the new phenomena is nuclear science. Introduced to a stunned world in its least attractive manifestation, nuclear science holds untold mysteries, unimaginable terror, and vast promises. Einstein said, when the atom was split everything changed except our thinking. Far too many people still think of thermo-nuclear bombs as super-slingshots Others realize that nuclear science might provide the vast reservoir of physical energy we need to produce abundance for all mankind. We

* Cyberculture is composed of "cybernetics," the science of control, and "culture," the way of life of a society.

have great hopes for atoms for peace and must search for a way to use atoms for people.

The second of the powerful new phenomena is not clearly focused yet, although a demonstration of the destructive potential of nuclear science has shocked the entire world to see, at least as a vague vision, the new concept: peace as a positive phenomenon, a valuable and workable instrument to settle human conflicts. That is quite different from man's past experience for since the beginning of history mankind has known as an alternative to war only the complement of war, an interlude between wars, occasionally even a reasonably prolonged absence of war. Even when there was no fighting, war has been regarded as the normal and accepted means to settle conflicts. Contrary to popular opinion, it is not a foregone conclusion that peace will bring about the millenium. There is no reason to believe that conflicts will disappear. And to use peace, rather than war, as an instrument to settle conflicts will require more ingenuity and intelligence and skill than to devise means to win wars. Difficult though it may be to live with, peace is the essential condition, if human civilization is to survive at all.

The least known and most far-reaching new phenomenon is the science of cybernetics and the revolutionary technology based upon its discoveries. Automatic systems and computing machines, even in their infancy, have an impact upon our world that could not have been imagined two decades ago; and they have the clearly foreseeable potential to produce not only unprecedented abundance for human beings, but relieve man forever of drudgery and toil. Yet, even experts still look at the computing machine as a super-abacus.

Any effort to deduce how observable phenomena are likely to develop and affect the environment involves some arbitrary assumptions that must be defined and granted. The major assumption in my hypothesis is so fundamental that, should it prove to be unreasonable, nothing on earth is likely to be proved or disproved again. I assume that the cold war will not be escalated into the nuclear fission of the earth, but that, on the contrary, it will continue to defrost. I further assume that all of us in the field of data processing and automation will continue to do our jobs with as much ingenuity and enthusiasm as we have in the past and to develop our precocious brainchildren, as we have every reason to expect from our auspicious start.

As we know, there is a great deal of confusion in the public mind about the words "automatic" and "automation," and about the effects of these rarely recognized phenomena. Economic pundits have made solemn pronouncements about the future impact of "automation" and based their predictions firmly upon a past experience with mechanization and its impact upon employment and the Gross National Product. A few months ago, Secretary of Labor Wirtz estimated that automatic systems have reached the intellectual level of human high-school graduates.[2]

Monumental fallacies are incorporated into such statements because the basic premises used by economic pundits and by the Secretary of Labor are incorrect; they confuse automation with sophisticated mechanization and use these basically incomparable phenomena interchangeably. If they could realize that the most sophisticated and efficient mechanical system—no matter how many electrical components are incorporated—is an open system that cannot operate unless the control loop is closed by a human being who must become part of the system, whereas an automatic system is a closed system in which the human component has been supplanted by a computing machine, they would understand that the conventional methods to inoculate the economy against periodic epidemics of unemployment and slackness are no longer relevant.

Before the Congress has been able to accept the Conventional Wisdom of one generation ago, everything changed. Everything but our thinking! And I must quote again John Kenneth Galbraith, who wrote that "the shortcomings of economics are not original error, but uncorrected obsolescence." We rightly cherish our intellectual heritage, but we must not allow it to calcify. The economic-political and social wisdom humanity may have acquired so painfully in the past must be tempered with new insights and forever re-evaluated with an open mind, just as the scientific and technical herit-

age of the past is constantly re-examined and re-vitalized by new discoveries and inventions.

II. *Agriculture and Cyberculture*

The present cybercultural revolution is comparable in magnitude only to the agricultural revolution, the ferment out of which all civilization arose. The agricultural revolution changed the earth from a jungle into a garden where food gatherers became food producers—who plant and harvest, who create a surplus over their needs and thus build civilizations. With the agricultural revolution man first began to emerge into humanity. He learned to control his environment, to adapt it to his needs, and to arrange his life into social patterns. The agricultural revolution that began to free man for his specifically human task changed the very nature of man.

Every society in the age of agriculture goes through recurring cycles of scarcity and surplus, of leisure and drudgery. For centuries this has been the human situation: part drudgery, part creative endeavor; part scarcity, part waste. The cybercultural revolution can create a world where machine systems produce undreamed-of abundance, and where human beings live human lives and are free to pursue as yet undefined human tasks.

Man in the Stone Age knew his task was to find food for himself and his young and to protect them from the dangers of a hostile world. He carved images on the walls of his cave, and sometimes there must have been a genius who observed the world closely, who somehow saw a pattern in remote incidents. He might have noticed that small plants grow into trees; that seeds spread by the wind or dropped by birds into the earth come forth again as plants, and that roots multiply and that some plants grew on the same spot again and again. He gathered the seeds and put them into the earth himself and watched over them and saw them bear fruit. And the age of agriculture could be born.

Man learned to till the earth to produce bountifully, to tame animals to help him pull the plow, to use the power of water and the wind to multiply the strength of his own muscles. In the course of many centuries man has developed complex tools which extend the perceptiveness of his senses and the skill of his hands, and devised powerful machines to extend the strength of his muscles. But man alone can direct and guide his aids. He must still labor for his bread.

The cybercultural revolution is brought about by the invention of devices that supplement the labor of man's mind. In the age of cyberculture the plows pull themselves, and the planting and harvesting is controlled by tirelessly efficient electronic slaves.

III. *What are Human Tasks?*

Man must learn to find new tasks to fill his days. If he no longer needs to pull the plow and clear the fields and forge the iron, how will he tire his muscles to earn his rest? How will he use his mind to earn his peace? How will he stand upon the earth he has not tilled in the sweat of his face, and feel that he is its master? What will he do with his life, if he no longer has to labor to earn his right to live? [3]

For centuries, and in every land, men have told stories about all-powerful, completely obedient slaves who would supply riches and ease. The brooms conjured up by the sorcerer's apprentice, the genie in the lamp, the monkey's paw—these are the stories of man's desire for a perfect slave and also of his fear. For man was always aware of his own inadequacy and he was not sure that he could control so perfect a servant with wisdom and with honor.

We can expect that in the age of cyberculture enormous populations will live in leisure. A few will "work." But no-one will labor in drudgery and sweat. This will be technologically feasible in a few decades. Invention can be speeded with the motivation for perfection. During World War II, the invention of radar was accelerated—in the opinion of eminent scientists—by many decades. But cultural lag may delay to bring cyberculture to its maturity for centuries. Reluctance to change obsolete ways of thinking, conflicts of interests, the shortsightedness of those who fear what they cannot fully understand can delay the future and use the best fruits of man's mind for his destruction rather than his joy.

IV. *The Problems of Transition*

The problems of transition from an agricultural-industrial to a cybercultural society are momentous. This is only the beginning. Unemployment, serious though it is, is not disastrously widespread yet. But soon it will be, if we refuse to face the fact that unemployment cannot be arrseted, even with the most phenomenal economic growth rate in the world, for the acceleration of automation will always exceed the acceleration of the growth rate. Unemployment must be changed to leisure. If we can learn to live with and use our electronic and mechanical slaves, rather than abuse our human bodies and our human minds, we can solve all the other problems that plague us now: the fear of unemployment, the envy the poor nations have for the rich nations and the fear the latter have of the former, the suspicious competition among the powerful. We negotiate about disarmament, but watching the unemployment figures rise, we quickly vote more money to be spent on producing lethal weapons. And as the unemployment monster rises, those who are gobbled up most easily—the unskilled—become afraid and rise in hatred and despair. Unskilled Negroes think it is the color of their skin that keeps them unemployed and white men fear that they will have to share the labor that is not fit for human beings and that none need to do in the age of cyberculture. Unions are losing members and try to stretch diminishing jobs by dividing them among more men, instead of enlisting as members those whose work can be done by machines and teaching them how to live human lives.

The slower the transition from an agricultural-industrial society to a cybercultural society, the greater is the suffering that must be endured, and the smaller the chances that—if humanity survives into the next century—the emerging age of cyberculture will be a good age for human beings. Slow transition does not cushion difficulties any more than pulling a tooth a little bit at a time softens the pain. The difficulties are not caused by the new age, but only by the transition itself—so that the problem can be solved only as transition is accomplished. *The best transition is a fast transition.* If we could have the wisdom to introduce as much automation as quickly as it is technologically feasible, we could create the age of cyberculture in two decades. Slow transition would bring such intense and widespread suffering that it may break into nuclear war—and end all civilization.

V. *Morality and Ethos*

To create the age of cyberculture requires something far more difficult than scientific discoveries and technical inventions. We must reexamine our moral values and our ethical concepts and the deeply ingrained notions to which we give lip service. And we must understand the difference between the moral values of mankind and the ethos of a society. The sanctity of human life, the worth and dignity of the individual are moral values that are absolute; these always have been true and always will be true, as long as there are human beings. But the ethos of a society is transient and it must alter with the needs of the society.

What we call our Protestant Ethic, although it is much older and spread far wider than protestantism, is the ethos of any society that knows scarcity and danger. It is a good ethos where virgin forests must be cleared, and wagon trains sent across a continent. It is a good ethos as long as men must wrest their meager fare from the earth with courage and fortitude and perseverance. In such a society, it is right that man should labor to plow the fields so that he might eat the fruits of the earth and bask in the sunshine of the heavens and dream under the shade of the trees. "Thou shalt eat thy bread in the sweat of thy face" is a good and reasonable precept in the age of agriculture.

Already the ethos of scarcity is becoming an unjust burden. All too often thrift is no longer a god, but the graven image of past days to which we give lip service. To save one's earnings and thriftily mend last year's coat, and use last year's car, and warm up last night's supper no longer is admired. But—the ethos that commands man to eat his bread in the sweat of his face still governs our personal lives and our national policies. Although for millions of human beings there is no place where they can put sweat on their faces, we still believe that there can never be another ethos for the future than the obsolete ethos of the past. And

every year we are condemning more than two million human beings to the swelling ranks of the unwanted. We suspect them of incompetence and laziness, or we pity them. We should re-examine the ethos that condemns millions who are simply the first contingent of citizens living under cybercultural conditions, without any preparation for the new age.

When human intelligence has invented plows that pull themselves, it is more virtuous to know how to play and to learn how to live for the joy of living than to bemoan the end of human toil.

As sons and daughters of puritans we do not know how to play and we look with terror at the "threat" of unemployment and idleness, because we can't conceive a promise of leisure. What we call play, recreation and entertainment, is not play, but its very antithesis. Play is something one does spontaneously, joyfully. We rarely do anything just for the joy of doing; but we do a great deal "in order to" gain something else. Instead of enjoying a holiday, we take a vacation—the very word signifies that it is merely a void between the activities we consider real. The "vacation" is something we use "in order to" have more strength for our labors. Recreation is something we pursue "in order to" re-create our energy. Entertainment is "in order to" forget our cares. We eat "in order to" replenish our energy. Our children are trained for the joyless ethos of scarcity and given candy "in order to" do something adults consider virtuous. Only the very young are fortunate enough to be ignorant of this grim purpose and suck their lollypops in blissful ignorance and joy. But even the youngest toddlers are not permitted to play for very long. Before they leave the cradle, they are but required to manipulate educational toys "in order to" learn control of their muscles or "in order to" learn to read. By the time they graduate from kindergarten we have infected our children and impressed them with our grim ethos. The joy of playing for the joy of playing is frowned upon. The joy of learning for the sake of learning has been destroyed by admonitions to learn "in order to" please mother, or to get good grades, or to get into Harvard or MIT twelve years hence. And by the time they arrive in Cambridge, they have not even the faintest memory of joy and play, and they grimly labor for their "credits," "in order to" graduate to obsolescent jobs.

VI. *Ethos for the Age of Cyberculture*

The proper ethos for the age of cyberculture is one that would serve humanity well to build a good society. We know so very little about living human lives in leisure and abundance, in dignity and self-respect, in privacy and the assurance of the fundamental human right to be unique as an individual. We confuse leisure with idleness, and abundance with waste. We view with suspicion the attempt of a human being to preserve his privacy and suspect it to be an attempt to hide evil. And we almost take for granted that an anomalism or eccentricity is necessarily inferior to conformity.

Nothing could be further from the truth! Idleness, like drudgery, is passive boredom suffered under duress, and waste is the misuse of anything—whether it is a scarce commodity or something plentiful. Leisure is the joyful activity of using our human potentials to the fullest, and abundance is intelligent economy, namely, the full *use* of natural resources for the good of human beings. Privacy is the fundamental right of civilized human beings and a necessity if one is to live harmoniously with one's fellow man. The uniqueness of individuals has made all human civilization possible; for the conformist cannot go forwards and only in the individualist's dreams and the dissenter's vision today can the reality of tomorrow be conceived.[3]

To learn to live in leisure and abundance is the task of this generation. Even if we wanted to, we would not have the power to choose between the past and the future. The cybercultural revolution cannot be reversed. But we can choose the future. *We* decide what kind of world we want to leave for our children; what *we* do now determines whether they shall exist in idleness or have a chance to live in leisure.

VII. *Early Signs*

Once we have grasped the fact that our present unemployment is only a beginning and that there can never again be a time when the labor of human beings will be required to produce what society wants, we can turn our human

intelligence to the problem of transition — namely, to prepare ourselves for the age of cyberculture by turning unemployment into leisure, by solving the transitional problems of scarcity, and by doing everything human ingenuity can devise to perfect our electronic slaves and complete all processes of automation.

We must rid ourselves of the erroneous idea that unemployment is still a negative period of waiting for a change to the positive state of being "gainfully" employed again. In this country, millions of human beings are in a *negative* state now. Many of them have been in this state for many months, years even, and many know that they will never be in any other state again. All the projections for the future—even the most alarming—consider only our past experiences. Only very recently have a few economists given their attention to the phenomenon of acceleration. "For too long they misled themselves and the public by projecting productivity into the future on the basis of the long-term average rate of past productivity gains. In so doing, they ignored the fact that their averages were a combination of relatively low rates in the distant past with significantly higher rates in more recent years."[4]

Computing machines and automation are barely in their infancy, and already our world has changed beyond all recognition and comparison. If we consider that all change is slow until it has overcome initial inertia, we can expect, before the end of this century, an increase in productivity that will dwarf the most alarming projections for unemployment. Solomon Fabricant, director of research of the National Bureau of Economic Research, warns that " . . . the long-term pace of advances in output per manhour has speeded up. It was 22 per cent per decade during the quarter-century preceding World War I. It has averaged 29 per cent since. During the most recent period —after World War II—national product per manhour has been rising at an even greater rate, 35 to 40 per cent per decade."[5] And to this should be added what is cautiously noted in the President's Manpower Report: "Although the statistical data on this subject are too limited to warrant definitive conclusions, it is probable that *underutilization* of plant, equipment, and manpower resources has had significant effect in retarding productivity gains since the mid-1950's."[6] Reuther concludes that "under the stimulus of automation and other revolutionary technologies, there can be no doubt that the historical tendency for productivity to move forward at an accelerating pace will continue into the foreseeable future."[7]

To the acceleration of technological advance we must add—or (more realistically) multiply —the acceleration in the rate of birth. The "war babies" and "post-war babies" will be flooding into the labor market—between 25 and 40 million of them in one decade. No rate of economic growth, no method of spreading jobs by decreasing the work week or extending vacations can absorb the enormous by accelerated flood of unemployment. Any dam or deflection that worked in the past—forced consumption, exploring underdeveloped continents or outer space, for example—cannot be used to counteract the potential power—for good or ill—of the increasing number and perfection of automatic systems that can produce 1,000 cars or 10,000 or 100,000 cars without human intervention and with—at most—a few human monitors to watch dials and stand by for rare emergencies.

If we allow human beings to remain unemployed because machines can do the drudgery of repetitive tasks, we are dooming untold millions to useless lives without hope and purpose. Even if we devise the means to feed them and supply them with the output of machines, they will not long remain in idleness and scarcity, while the products of machines rot in warehouses.[8]

VIII. *Lessons of History*

Instead of dooming the vast majority of mankind to idleness and unemployment and the indignity of the dole, we must prepare now for leisure and abundance. There are some lessons we can learn from history. In the Golden Age of Greece we can study a society of leisure and abundance based upon wealth that was not created by the labor of any of the members of the society, but by slaves.

We piously deplore the evils of obsolete slavery and believe it right and proper to condemn millions to starvation, or, at best, the indignity of the dole. Let us look at Greek society honestly and examine how an unsurpassed

civilization was created amidst the wealth and leisure which, twenty-five centuries later, might well have been produced by electronic and mechanical, instead of human, slaves.

The Greeks differentiated clearly between the private life of a human being his life in his household which produced the necessities—*oika,* the Greek word for "home," is the root-word of economics—and his life as a citizen, which Aristotle called *bios politikos.* The "good life" was the life as a citizen, was "good" because man, freed from labor by having mastered in his household the necessities of life, could pursue human tasks. "At the root of Greek political consciousness we find an unequalled clarity and articulateness in drawing this distinction. No activity that served only the purpose of making a living, of sustaining only the life process, was permitted to enter the political realm, and this at the grave risk of abandoning trade and manufacture to the industriousness of slaves and foreigners writes Hannah Arendt."[9]

However we may deplore the private, or household, life of the Athenian—in this century of electronic slaves we can so easily afford to condemn human slavery—we can only admire the unequalled height of civilization his public life produced. In his public life every Athenian strove to excel, i.e., to distinguish himself from all others, to be a unique human being, an individual unlike any other that ever lived or ever will live. The Athenian lived a human life, in play and work, but never in drudgery and labor. "Who could achieve well if he labors?" asked Pindar.[10]

Several hundred years later and several hundred miles to the west of Athens another society existed whose citizens were freed from the necessity of labor in order to sustain life. But whereas freedom from want and the necessity to labor emancipated the Athenian into a human being who achieved excellence, Roman citizens became an idle mob under equivalent conditions of affluence. The decline and fall of the Roman Empire, wrote Edward Gibbon, is "the greatest, perhaps, and most awful scene in the history of mankind. The various causes and progressive effects are connected with many of the events most interesting in human annals: the artful policy of the Caesars, who long maintained the name and image of a free republic; the disorders of military despotism . . ."[11]

The essential difference between Greece and Rome is the difference in their points of view, in their ethical concepts. The Greeks strove for individual execellence; they wanted to create beauty and contemplate the mysteries of the universe. Abstraction and generalization were their inestimable contributions to science. The practical they dismissed as not worthy of discussion and recording. Archimedes, whose practical inventions covered an astouding variety of applications, never thought them worthy of description. He wrote only about abstract mathematics; we learned from his Roman enemies that he invented marvelous machinery.

The death of Archimedes, by the hand of a Roman soldier; as the great mathematician stood contemplating a diagram he had drawn in the sand, is symbolic of the end of an era. The Romans were great organizers, "but," said Whitehead, "they were cursed by the sterility which waits upon practicality. They were not dreamers enough to arrive at new points of view."[12] No Roman ever lost his life because he was contemplating abstract mathematics!

Rome, her unemployed citizens idly seeking *panem et circenses,* destroyed herself. The moral disintegration of Rome had begun long before Christ was born. Her conquests brought Rome only material luxury and human proverty. Roman citizens received their dole and idled away their humanity in ever more brutal titillation. "It was because Rome was already dying that Christianity grew so rapidly. Men lost faith in the state, not because Christianity held them aloof, but because the state defended wealth against poverty, fought to capture slaves . . . they turned from Caesar preaching war to Christ preaching peace, from incredible brutality to unprecedented charity, from a life without hope and dignity to a faith that consoled their poverty and honored their humanity. . . . The political causes of decay were rooted in one fact—that increasing despotism destroyed the citizen's civic sense and dried up statesmanship at it source. Powerless to express his political will except by violence, the Roman lost interest in government and became

absorbed in his business, his amusements, his legion, or his individual salvation."[13]

We might ask why despotism increased in Rome, why the Athenian sought his excellence in art and philosophy and science, and the Roman in the material luxuries that were all he gained from his conquests. We might ask why Christianity so very quickly forgot that Christ taught human beings to live for the Glory of God, which means to live for the joy of living, of being human and why the ethos of scarcity perverted "living for the Glory of God" into laboring "in order to" assure the glory of the church. We might ask why the Athenian, though conquered and enslaved, mastered even his enslavement and his conquerors. We might ask whether the Golden Age of Athens could have endured if the Athenian had found a way out of his dilemma: his need for leisure and his rejection of human slavery.

Returning to our own century of transition, we can rejoice that we have what humanity never knew before—slaves to free us from the necessity of laboring "in order to" sustain life that are not human, so that we need not be ashamed to enjoy what they produce. For the first time in human history, man can be free. Machine systems can provide him with leisure and abundance, and rescue him from the degradation of being either a slave or a master of another human being.

But machine systems can do only what man wants. If human beings cannot learn to distinguish between human tasks and toil fit only for machines, if we persist in competing with the machine for the repetitive, dreary, stultifying, de-humanizing jobs for which only machines are suited, then humanity wlil become enslaved by the machine more cruelly than it has ever been enslaved by any despot of the past. For the machine provides us with slave labor; and, therefore, human beings who compete with the machine are, thereby, accepting the conditions of slave labor. Human beings who learn to use the machine wisely, on the other hand, will be freed by the machine to achieve excellence.

We are at the cross-roads: one way leads to the Athens the Athenians could only dream of; the other to a Rome more dreadful than the most ghastly Roman nightmare.

Greece or Rome—that is the choice we have, the choice we must make now, the choice we should have made yesterday and for which tomorrow will be too late.

PART TWO—METHODS OF TRANSITION

I. *Educating the Young*

A practical and relatively painless method to accomplish the transition into the Age of Cyberculture must begin with the education of the young. We are well aware of the fact that unemployed youth has already become a social problem, and we know that what we so inadequately call "juvenile delinquency" is not restricted to the underprivileged. The violence of youth and the crimes committed by children show, of course, the general moral decline. Even more serious than isolated outbreaks of violence, even more desperate than gangs of destructive hoodlums, is the widespread indifference and bewilderment among the young—whether they stay in schools that provide nothing but bland custodial care or whether they are drop-outs.

The real problem of the young is that there seems to be no place for them in the world. They know society looks with dread upon the vast numbers that are pouring out of schools, and they know that it is wrong for them to be met with fear and loathing. They are the future of mankind, and they have a right to be welcomed with joy.

What would happen if the 25 to 40 million young people who will pour out of our inadequate school system in the next decade were not to flood an already overflowing labor market, but enter instead into a period of basic education for the Age of Cyberculture?

It would be infinitely harder—perhaps impossible—to change very profoundly the prejudices of those who have learned to labor and who have labored for too long. If their labor is taken over by machines, we can only make their emancipation which came too late for most to enjoy in leisure, as pleasant and comfortable as society can afford. And we can try to make their idleness not too shameful a thing.

But for the young we must do far more than train them to become another obsolete generation of laborers, for there can be no honest labor for them and no dignity in toil. Any human being who seeks to labor in competition with the machine is doomed to slavery and to the conditions of slavery. There is no human ditchdigger who can live on a scale low enough to let him compete with the steamshovel, and there is no human bookkeeper, or mathematician, who can compete with a computing machine.

There is no human printer who can compete with a tape-fed printing press. And there is no human metal cutter who can compete with computing-machine controlled machine tools. The keyword is "compete." We can no longer afford to measure the value of a human being in the market place.[14]

In 1955, when the A.F.L. and the C.I.O. consummated their marriage of convenience, President Meany promised that the newly-weds would become parents of an expanding family; they would "organize the unorganized." But with the sole exception of Hoffa's teamsters—who are, at best, considered naughty stepchildren—the family has not proliferated. The auto workers have lost 300,000 members since 1953—and this in spite of the fact that the industry has achieved a glittering production record in 1963. The steel workers have diminished by 250,000. There are half a million fewer mine workers, and 760,000 fewer railroad workers.[15]

In spite of all the efforts made by labor unions to spread the work and to delay the dismissal of workers, it takes considerably fewer of us to produce considerably more. Whether we regard feather-bedding an evil or a necessity—it does not prevent the spread of unemployment. At best it delays the inevitable disaster for a few; at worst it retards important improvements.

The ranks of labor are diminished by the fired, but even more significantly they are starved at the source by the vast numbers of "the unhired." Among the ever increasing number of the unhired, labor unions must find new blood and new strength, and a new lease on life. Labor must forget the organization methods of the past, when "marginal workers" were considered a poor investment. Many old-time union men say "they bring not back in dues what it costs to organize them," and are thereby guilty not only of greater callousness than that for which they blame management, but of irresponsible short-sightedness. Unions who do this sort of cost accounting while they invest their estimated union wealth of $1.5 billion in blue-chip securities and profitable real estate, are doomed to die of their own corruption and decay.

Labor and management must learn to invest in human beings. If labor unions would "organize" youngsters who graduate (or drop out) of an antiquated school system, their membership rolls would swell (and without any crippling diminishment of their coffers) and their vigor would be restored. They would once again have a vital role to play in the society. They would once again breathe the fresh air of the future instead of suffocating in bank vaults clipping their coupons.

Organized Labor should offer to educate every boy and girl who wants an education for the Age of Cyberculture. It is to be hoped that colleges will, in time, adjust their curricula to the needs of the future. With a few exceptions our institutions of "higher" learning are custodial, rather than educational, and perpetuate training their unfortunate students for obsolescence. Education must not be equated to training for obsolescent jobs. Since our present feeble attempts at "re-education" are not educational at all and do nothing at all to prepare human being to live in leinsure and abundance, they could not, and do not, have the slightest effect upon our present unemployment problem. Such "re-training" efforts are like aspirin; it can disguise a headache for a while, but it cannot cure cancer, or even a headache.

We throw a feather into the Grand Canyon and we are surprised that there is no echo! The education that must be provided to get an echo from the age of cyberculture must make it possible for human beings to learn how to live human lives and to create an ethical system that will permit human beings to do whatever they do gladly and for the sake of the thing itself; and not reluctantly and only "in order to" make a living.

A good curriculum might well start with questions about Greek civilization and Roman decay. Whatever the specific subject, its aim must be to open the eyes of our twentieth-century blind children to the eternal miracle of life.

One labor leader, enlightened about the vital need to educate human beings, says: "America will be a much better place when everybody works four hours a day and attends some kind of classes four hours." He made a start—small, but of tremendous significance. His local has financed scholarships for children of members. More important, it is seeking out latent talent among its members (in the hope of developing the union's future leaders) for the "Futurian Society." The best-educated among the members conduct courses for other members. And at the local's Long Island estate, seminars are held in such "impractical" subjects as literature and art and philosophy.[16]

Since the education of several millions of youngsters involves far more than a one-week seminar, much more is required than a Long Island estate and the funds a local can afford. Unions are not *that* rich. But humanity is. Such a vast education program must be financed by government subsidy, in part. This can be justified—even to the satisfaction of the victims of the Puritan Ethos—as a perfectly reasonable investment which, partially at least, pays for itself out of savings in unemployment compensation, relief, and the costs of custodial care for those who would surely commit crimes if they cannot find a positive purpose in life.

Another source of financing should be supplied by the very machines who have replaced human laborers. At least one manufacturer of automatic machines is a poineer in this approach. He "taxes" every machine he sells by putting away a certain sum which contributes to the support of a foundation to make various studies of several facets of the problems created by the very existence of automatic machines. "If machines perform our labor, then machines will also have to pay our taxes."[17] This may be a socially acceptable way of saying that the abundance produced by machines must be available to human beings, lest it rots away and destroys humanity with its fetid decay.

Union funds, special taxes paid by manufacturers, a contribution made by the entire population in the form of government funds are three sources to finance the preparation of youth for the Age of Cyberculture. Of course, it would be foolish to prepare millions of young people for a world which, simultaneously, we try to postpone. When we no longer need to be concerned about new floods in the labor market, there is no longer any reason to attempt any delays in complete automation. We are then free to encourage technological inventiveness and to complete the process as quickly as possible, not only in the industrialized nations but—under the sponsorship and tutelage of the United Nations—of the underdeveloped countries.

Such an acceleration of automation would insure full employment for the existing labor force, not only in feather-bedding and busywork, but by the full occupation of highly trained personnel. It is the only method of achieving full employment *and* full occupation. We have neither one nor the other now.

How many scientists and engineers are still employed, but under-occupied? How many pass their days in idleness at their well-designed desks and in their well-equipped laboratories? How many repeat endlessly insignificant experiments, and waste all their ingenuity in inventing more innocuous re-search (not research) projects?[18]

II. *Experiment in Attrition*

It is highly probable that, without the influx of youngsters into the existing labor force, normal aging and retirement—a process called by modern economists and labor experts "attrition"—will diminish the labor force over the next two decades at approximately the same rate as machines replace human drudgery. In the pioneering agreement, the "Long-Range Sharing Plan," the union and the Kaiser Steel Company pointed the way for a company and its employers to share the fruits of automation. Whatever can be saved by greater productivity and efficiently is divided: one third to the workers, two thirds to the company (which must share its two-third's portion with the government).

Any laborer whose job is eliminated by a machine continues to draw his pay for one year. During this time he is placed into a labor pool which acts as a reserve to fill in for absentees.[19]

It is too early to draw general conclusions from the Kaiser experiment, but the officials of the steel workers' union are reasonably pleased and the company considers the pool an asset. This plan does not provide a complete answer to the problems of automation, and it does not stop the process of replacing human drudgery with machine slaves. It does provide a cushion for a few individuals.

III. *Hope For the Future*

If one steel plant can create a labor pool, if one manufacturer can tax his machines and use the money to encourage study, if one union local can provide a place for learning and reflection for its members with such remarkable results, we have good reason to be hopeful for a bright future.

If we set ourselves as a long-range goal a good cybercultural society, we can solve the intermediate problems and devise appropriate measures to overcome the immediate difficulties that are attributable to the phase of transition rather than the advances in technology. The immediate consequences of diverting the young and unskilled from the labor pool and into a constructive program of education for the age of cyberculture would be dramatic. It would, first of all and for all times, wipe out the demoralizing condition of hopeless unemployment.

We know that in this rich country there is considerable and stubborn poverty. Why does it exist when granaries are bursting with surplus food, when farmers are paid for consenting to let their fields lie fallow, and when stores are filled with every sort of consumer product so that customers must be enticed to want what they do not need? We know that poverty, in this country, is poverty in spite of abundance—poverty caused by inadequate means of distribution, by the ethos, not the real existence, of scarcity, i.e., by unemployment. To wipe out unemployment and the ethos of scarcity is to wipe out poverty.

This is not true of poverty everywhere on earth as yet. There is still real poverty in this world, and there are still poor nations, although even the poorest nation in the age of cyberculture is not intrinsically and forever doomed to poverty or the charity of others. That is why we call the poor nations quite appropriately underdeveloped nations. The most enlightened domestic policy we might pursue cannot solve world-wide problems, and the most intelligent and ingenious international agreements cannot forever eliminate the danger of nuclear war. We must do much more. We must not have underdeveloped nations. The age of cyberculture must be universal.

IV. *World-Wide Cyberculture*

If we eliminate unemployment in this country immediately and proceed to create automation at the fastest possible rate, we shall almost simultaneously free an enormous number of highly trained and skilled people—the employed but under-occupied—who could, under the sponsorship of the United Nations, assist and advise the underdeveloped nations to build modern automated industries.

At the present time, most of the newly formed nations of Africa seem to be diligently creating nineteenth-century conditions of the worst sort. It is not surprising, since most of their leaders were educated in Europe or the United States two or three decades ago so that they were imbued with the ideas of nineteenth-century Europe and America.

They dream of leading their countrymen out of the jungle, and they are bringing them straight to the horror of nineteenth-century city slums. It is preposterous to lead human beings out of insect-infested green jungles and turn them into obsolescent masses of unskilled and unwanted laborers in rat-infested city jungles.

Surely among all the remarkable intellects that have asserted themselves in the new nations of the world there must be some who can understand not only John Locke and Voltaire and Marx, but also Russell's mathematics and logic and philosophy, and Wiener's cybernetics. Surely there must be some among them with the imagination *not* to imitate nineteenth-cen-

tury American-European industrialism, but to create twenty-first-century cyberculture. Such a person, if he also has power in his country and influence among his people, can conserve the most admirable values of his native culture and create, at the same time, good living conditions for his people. He could bring to his country the best our European-American civilization has to offer without dooming them also to the worst.

If the ethos of the society that sees virtue in laboring "in order to" gain something—bread or status, or self-respect—changes to an ethos of abundance, it will be virtuous for human beings to live human lives in leisure and abundance. And no-one can be sure what undreamed-of heights humanity can reach when human talent is no longer wasted in the basic struggle to survive.

It is surely the only sensible and practical choice to prefer leisure and abundance to idleness and waste. The choice must be made—and it must be made now. We cannot ignore the powerful new phenomena human intelligence has created. We cannot abdicate our responsibility to choose how they shall be used. For in our very abdication of choice, we would choose the worst alternative: to drift blindly toward disaster.

If we want to conserve our traditional values —the right of the individual to life, liberty, and the pursuit of happiness—we must choose wisely and act boldly.

Our history was made by human beings with bold vision and good sense, with deep moral convictions and human compassion for human frailty, with respect for the dignity of human beings and love for mankind, with the imagination to dream and the courage to act. Such men and women cross oceans, transform continents, and build the City of Man!

That is our heritage.

Appendix

A Program for Transition—Immediately Attainable Goal

Keeping long-range goals clearly in mind, we must make many choices and decisions about measures to relieve the problems caused by transition. Cybernation affects the unskilled earlier than the skilled, it makes certain skills obsolete sooner than others. This causes untenable conditions of abject poverty and a sense of personal failure in the midst of great affluence and achievement. It takes time to erase the prejudices and superstitions accumulated for centuries.

The following suggestions to make the transition period as constructive as possible and to prevent suffering is a goal that is attainable. None of the measures proposed will have an adverse effect upon the long-range goal of creating a cybercultural society where human beings can live human lives. And, to some extent, all measures will contribute to achieving a cybercultural society as soon as possible and in a humane manner that is of benefit to the individual.

I. *Government Action*

1. An *agency for cyberculture* should be set up. The agency must have *sufficient funds and power* to study the over-all ramifications of new phenomena, to study specific recommendations made outside the agency, *to initiate large-scale experiments,* and *to carry out programs* the agency wishes to adopt *without undue delays* imposed by the Congress or other governmental and nongovernmental bodies. The Agency should have representative of all branches of government, of organized labor, of industrial management, of academic life, and—most important—of the unorganized consumers. Representatives of the unorganized consumers, i.e., the vast majority of citizens, should be knowledgeable people who are completely distinterested, but intensely committed to the goal of a good cybercultural society with the courage to try the untried and the reverence for permanent values that must be conserved. A test laboratory should be set up immediately (Long Island would be most suitable).
2. *Public works:* there is an almost inexhaustible need for an extensive program of public works, such as road building, construction of dams, irrigation, conversion of

sea water, construction of hospitals, schools, parks, recreation and holiday facilities, beautification of cities, slum clearance, low-cost (but comfortable and beautiful) housing (about one million units per year is perfectly attainable), a reasonable power system to provide abundant and low-cost power for industry and homes in the most remote sections of the country. In addition to an extensive public-works program, subsidies to individuals should be provided, on a generous scale and in generous amounts, to artists, writers and other intellectuals, and to artisans and craftsmen. This will encourage the most valuable members of a society and make it possible for them to devote all their energies to their human tasks; creating a great civilization with a blossoming of art, science, and philosophy, as well as a great revival handcrafts. Pride in artistic and intellectual achievement and pride in fine craftsmanship would benefit not only the individuals subsidized but the community at large.

3. *Constitutional guarantee of Living Certificate.* All means of distributing funds to those who cannot find jobs should be coordinated. This would make it possible for the most severely (financially) handicapped to move from congested and expensive areas. Most of all, it would help to restore a sense of dignity and worth to the unemployed and make it possible for them to learn the use of leisure.
4. *Generous Re-location Allowances.* Financial assistance for relocation would make it possible for families to move to an area where (1) *jobs* might be available, (2) *educational facilities* might be accessible, and (3) *climatic and housing* conditions might be advantageous. Although reasonable precautions to prevent undue abuse may be required allowances *should not be restricted* to job opportunities.
5. *Large-scale education program* to assist the chronically (often for generations) *under-educated,* to *develop the potential of the gifted* by enabling and motivating them to continue their schooling for as long as they can benefit thereby. Education facilities should be extended and made practical for *adults* up to any age when there can be a reasonable expectation of achievement. (Achievement to be defined in terms of a personal sense of dignity rather than material success).
7. *Rehabilitation of neglected and deteriorating areas:* urban and remote on a generous scale and with the conscious intent to *restore natural beauty and create and/or restore urban beauty.*
8. *Transportation: construction of rapid-transit systems,* both urban and inter-urban, with particular regard for the comfort and convenience of passengers and the beauty of the community.
9. Study of the *best use of the licensing power of government,* and exercise of the best use of such power, to carry out the transition to a cybercultural society as rapidly as possible.
10. *Use of the taxation power of government to expedite* and ease the transition from human labor to automatic machine systems, apportioning costs of the transition period fairly, motivating industry to cybernate and finance the generous application of the principle of constitutional Living Certificates.
11. In the process of *conversion* from a defense-oriented to a peace-time economy, all *obsolete military installations and materiel should be transferred to the community for the best civilian use.* Transfer should be guided; trained personnel (from the military) could participate in the conversion to civilian use.
12. A *large-scale education and public relations program to explain the cybercultural revolution* should be devised and carried out with the cooperation of non-governmental organizations.

II. *Organized Labor*

1. The labor movement should play a significant role in the transition period by recognizing that their *responsibility extends to the unemployed,* as well as to the (as yet) employed.

2. *Organizing the unorganized* including young people leaving school and unable to find employment.
3. *Vast educational programs to prepare union members* (and others) *for leisure.* Such programs should be *non-utilitarian* and designed primarily to stimulate people to think through the familiar patterns of the past and distinguish between obsolete conventions and permanent values. A realistic program designed to make "free" time a time of leisure and accomplishment rather than idleness would include education for hobbies and, simultaneously, foster an awareness that the ethos of scarcity is not necessarily virtuous.
4. *Investing the vast welfare and pension funds* of unions in enterprises of social value and thus exerting enormous influence upon the shape of our economy and our value system.
5. *Cooperating in all programs to expedite the transition* to a cybercultural society and simultaneously entending the sphere of influence of organized labor to those whose laboring has been taken over by machine systems.

III. *Individual Action*

1. In every kind of *community, professional, special-interest or other types of organizations, the individual can spread his insights about the cybercultural revolution.* Discussion with friends, with large groups can serve to alert others to the fact that changes *are* taking place, that changes are not necessarily to be feared, and that we do have choices to make to determine the kind of world that is to evolve from the changes. For although we can not elminate the fact of change, *we can determine the direction into which change propels us.*
2. The thoughtful individual must *seek as much information as possible,* sift information carefully, and *articulate* what he has learned.
3. The *education of the population in a free society is the concern and responsibility of all citizens.* Education can spread by individual and group action.
4. *Interest and concern for the action of government, industry, and labor, is the duty of the individual.* No government program is too complex, no corporation too rich, nor labor union too powerful for most careful scrutiny by the individual. No injustice is too minute, no human suffering too remote, no person's fate too unimportant to affect profoundly the structure of the society. If the most insignificant right of the least important among us is endangered ever so slightly every right of every one of us is in danger. Only the individual can guard the rights of the individual. And the individual can and must demand that all action be carried out for the benefit of individuals in the society. It is still true that *a nation has the government and institutions it deserves.*
5. Individuals, singly or through their organizations, must *demand generous educational facilities and sources of information.* The media of public information should be re-organized to *serve the public interest. Freedom from commercials on the air and from page after page of advertisting obscuring information in printed material should and can be demanded by individuals* (which does not exclude the use of government licensing power to protect the interest of the individual.)
6. Individuals should make conscious efforts to *examine their own values, discard their prejudices, and accept new ideas.* "The unexamined life is not worth living," said Socrates. This clearly makes it the responsibility of every citizen to examine his own life and to assist others (though not force it upon them, because that would be ineffective) to examine their lives.

CITED REFERENCES AND BIBLIOGRAPHY

1. JOHN KENNETH GALBRAITH, *The Affluent Society,* Houghton Mifflin Company, Boston (1958).

2. Secretary of Labor, W. W. WIRTZ, Address to the National Convention of the A.F.L.-C.I.O., New York (1963).
3. ALICE MARY HILTON, *The Age of Cyberculture, A Series,* The World Publishing Company, New York (1963).
4. WALTER REUTHER, Statement for the Senate Subcommittee on Employment and Manpower (May 22, 1963).
5. SOLOMON FABRICANT, Director of Research, National Bureau of Economic Research, Annual Report (1959).
6. President's Manpower Report (1962).
7. WALTER REUTHER, Statement for the Senate Subcommittee on Employment and Manpower (May 22, 1963).
8. ALICE MARY HILTON, "Computing Machines: Curse or Blessing," *The Age of Cyberculture,* Syndicated Series, North American Newspaper Alliance (August 11, 1963).
9. HANNAH ARENDT, *The Human Condition,* University of Chicago Press, Chicago (1958).
10. PINDAR, *Carmina Olympica,* xi. 4.
11. EDWARD GIBBON, *The Decline and Fall of the Roman Empire,* vol. I - III.
12. ALFRED NORTH WHITEHEAD, *The Aims of Education.*
13. WILL DURANT, *Caesar and Christ; The Story of Civilization,* vol. 3, Simon and Schuster, New York (1944).
14. ALICE MARY HILTON, *Logic, Computing Machines, and Automation,* Meridian Books, New York (1964).
15. ALICE MARY HILTON, "Cybercultural Revolution," *The Minority of One* (October 1963).
16. HARRY VAN ARSDALE, JR., Report to the American Foundation on Automation and Employment, New York (1963).
17. JOHN S. SNYDER, Report to the American Foundation on Automation and Employment, New York (1963).
18. ALICE MARY HILTON, "Automation Without Tears," *Electro-Technology* (August 1960).
19. DAVID COLE, Report of the Kaiser Steel Agreement of March 7, 1963, Washington (1963).

INFORMATION PROCESSING AND SOME IMPLICATIONS FOR MORE EFFECTIVE MANPOWER PROGRAMS

Herbert E. Striner
The W. E. Upjohn Institute for Employment Research
Washington, D. C.

This brief paper will focus on the tremendous potential which now exists for the development of a more rational and effective technique for dealing with problems of manpower utilization. This topic should call for a far more detailed and sophisticated treatment than it is about to receive from me. For this, I must apologize, both to this audience and my own conscience. I feel, however, that the opportunity should be taken to outline to this particular audience the sort of information processing system which I feel must be developed if we are to have a more effective means for dealing with the growing manpower problem in the U.S.

In discussing the implications of information processing for manpower problems, what I am really concerned with are the two major problems, *unemployment* and *underemployment*. At the present time, our nation would appear to have an embarrassment of riches in both of these areas. At the rate we are going, I'm somewhat pessimistic about our becoming less wealthy with respect to these two surpluses. Aside from the fact that insufficient levels of demand for goods and services necessary to increase the demand for labor would appear to be on the horizon, there is an increasingly difficult problem of an inadequate amount of information to guide both potential employers and employees in the match-making process we call the "labor market." Before indicating the deficiencies of our labor market and what the implications of information processing may be, let me briefly describe what the economist sees as the functions of the market.

One of the major assumptions of economic theory is that of the "market place" and its function of establishing contact between the forces of supply and demand. In the market place those with a product or service to sell meet those who may be interested in the purchase of the product or service, and in the bargaining process establish the "market price" and consummate the economic transaction. In this way, the needs of the consumer and the products of the suppliers lead to the pricing and production decision policies which ensure that the economy is functioning in an optimal manner—at least with respect to supplying what the market indicates is needed.

Obviously, I have not yet included a factor of major importance regarding the function of the market—that of communication. In the case of many products and services, we are all aware of the consummate and skillful use of the various news media in apprising us of their qualities. There is not only very little of a lag between the entrance of new products into the market and the public information through television, newspapers and magazines of these new items—but many of us are very frequently informed months ahead of the imminent introduction into the market of the new, completely revolutionary "widget" without which no home, car, person, or society can be complete. Indeed, we are

even made aware of those types of items which are available for the person "who has everything."

But in one all-important sector of our economic life, the paucity of information and the turtle-like movement in the direction of the development of a communication system to connect the forces of supply and demand represents what is tantamount to an unpardonable sin. I refer to our labor market and the theoretical mechanism for matching available and projected job needs with the unemployed or those whom we can foresee as being unemployed.

As you undoubtedly know, the unemployment situation in the United States has been one of increasing concern since the mid-1950's. In the past 10 years, we have not been at a level of unemployment which even the most conservative economists can view with equanimity. Though the present rate of 5.6 per cent is far above the 3 per cent we usually view as the maximum desirable level, there are many economists who feel that the real rate of unemployment is much higher. The bulk of this is concentrated among the young, the old, and the minorities—chiefly the negroes. There is currently raging an argument between economists as to the reasons for this rate of unemployment, which, in an unprecedented manner, continues to hang on like a summer cold during the sunny seasons of apparent prosperity. Some claim this results from structural changes in our economy which throw large numbers of people into the category of the unemployable, while the other group contends that the real culprit is too low a level of economic growth. For our purposes today, which is the culprit, or whether it is both, as I believe, has relatively little significance. In either situation, there is an increasingly major role which computers must begin to play if we are to move in a direction of an optimal utilization of manpower. Several factors would lend a sense of the imperative to the application of a more scientific system for the acquisition, storage, retrieval and dissemination of labor market information.

Increasing rates of technological change produce a shorter lead time between the potential of the innovation and the innovation itself and its possible resulting labor displacement. Consequently, the amount of time in which society can anticipate retraining needs and methods for increasing worker mobility is shortened severely. With an ever-increasing labor force, the numbers are far from inconsequential. I might also add that along with the obvious economic facet of the unemployment problem, given the age and minority group characteristics of this unemployment population, the stimulus for dealing with this problem in a more logical manner would seem to increase rapidly.

How can computers be of importance in dealing with this situation? To begin with, computers and the development of a labor market communication system will not *create* jobs, at least not in the numbers necessary to put a serious dent in the unemployment population. But a communication system can begin to provide a much more effective means of establishing linkages between those possessing specific skills and those possessing the jobs for which these skills are needed. Those of us who have been involved in labor market analysis are more and more aware of jobs for which there are no takers because the takers are unaware of the job openings. Logically, one asks, what about the use of such agents of communication as the newspapers, private employment agencies and the United States Employment Service. In examining these major possibilities, several "necessaries" for an optimal solution should be kept in mind. To begin with, the description of the available job and the matching skill must be fairly detailed; second, there must be a formalized network which brings together the parties most interested logically in each other's potential; third, there must be the maximum coverage by industry and geographical area, of all job and skill-matching potential; and fourth, there must be a minimal delay in sending the message to the logical parties, both on the supply, or employee, side and the demand, or employer side.

Now what is the situation with respect to newspapers, private employment agencies, and the U.S. Employment Service? If we begin with newspapers, I think we immediately see that on the employer's side, there is a real opportunity for advertising in ample description. With regard to the unemployed individual, however, it is difficult to envision a situation where the

individuals who have been laid off, many of whom are in the unskilled or semi-skilled categories, will be in a position to advertise. This is so, firstly, because of the financial problem. With very limited financial means it is impossible for an unemployed individual to be able to advertise in sufficient length so that an adequate background description can be given. Aside from this, many of the items and characteristics which we have mentioned above could not possibly be contained in the advertisement which an unemployed individual might conceivably place in a newspaper. Secondly, the newspaper is hardly a formalized network which can bring together the parties most interested logically in the potential of each other; that is, the potential employer and the unemployed individual who may have the characteristics being sought by the employer. It is a very informal source of information with no compulsory or systematic feature about it. As a matter of fact, in a study done in 1962 by Eva Mueller and Jay Schmiedeskamp, supported by funds from the Upjohn Institute, it was found that only between 5 and 10 per cent of the unemployed who were able to locate new jobs had been able to locate the jobs as a result of their use of newspaper advertisements. The newspaper very seriously lacks the potential of a network which can be institutionalized and which can continuously and in an exhaustive manner bring together all of the sources of information which we need for the network system we must begin to envision. Since it is also completely voluntary, there is no real opportunity for an extensive coverage either by industry or by geographical area.

If we look at the private employment agencies, we immediately see that there is the major hurdle for many of the unemployed of a substantial fee or part of their salary which would be deducted upon placement. Private employment agencies also lack the tremendous scope of coverage in terms of industry and geographical areas which we have set forth as one of the major criteria for a successful system. One of the objectives which the system we are beginning to formulate should have is that of being capable of directing the unemployed to positions available outside of the immediate areas in which they reside. Another problem of the private employment agency is that very frequently they tend to choose those sorts of individuals whose skills represent a higher probability of placement rather than a lower probability of placement. The very practical reason for this has to do with the high cost of continuing to attempt to place individuals whose skills are such that they represent a higher cost rather than a lower cost problem with respect to placement. The job placement success of employment agencies can be commented upon quickly by alluding to the Mueller-Schiedeskamp study, where it was found that only between 7 and 14 per cent of those unemployed who finally found work found the job as a result of the use of an employment agency, public or private.

Finally, let us look at the United States Employment Service. The job description information and the matching skill information developed by the U.S.E.S. may or may not be fairly detailed, depending on the efficiency or the employment service officer or interviewing personnel in the individual state agencies which make up the U.S. Employment Service. Like private employment agencies, between the states we have an erratic behavior pattern with respect to the adequacy of the job description information. However, the U.S. Employment Service does have the skeleton of a formalized network which can bring together parties most interested in each other's potential. This is so because it is organized with a central focus coming from the Federal Government with employment services located in and controlled by the individual states. This is a very important potential with respect to our envisioned communication system. By virtue of this distribution of officers, there is the potential coverage of each of the industries and geographical areas in the United States. However, the U.S. Employment Service, like the other sources of information, lacks in the ability to have a minimal delay in sending messages to the logical parties both on the supply—or employee—side and on the demand—or employer—side. Of great importance, however, is the fact that within the present unemployment placement services of the U.S. Employment Services, there is the seed from which can grow a major automated computer operations system which is calculated to provide the basis for matching information on

available jobs with available individuals for these jobs. Under the original Wagner-Peyser Act, which authorized the establishment of the U.S. Employment Service, there was authorized a system of job-clearances between the states, and during World War II this activity was of tremendous importance in achieving a greater degree of mobility of the labor force and moving scarce skills to defense industries. This activity has continued as part of the employment service and has been recently improved. Its most effective recent operation has to do with the professional office network. This particular plan deals with making potential jobs available to professional people and potential workers available for the consideration of employers needing their services on a nationwide basis. This program was begun in March of 1956 after a number of pilot projects and experiments in 8 states and the District of Columbia and Puerto Rico. At the outset, the plan was to provide for a flow of unfilled orders and unmatched applications for the local office to a central "key city" office in each one of the individual states concerned with attempting a state-wide effort to find openings or recruit applicants. Beyond that there was an exchange of still unmatched orders and applications among the various key cities which were defined. This was expanded so that by 1963 there were 121 professional network offices located in key parts of all sections of the United States. This precedence could perform an important function in that it begins to indicate a prototype on the basis of which a larger automated massive acquisition, storage, retrieval and dissemination system could be developed. As I would envision the system toward which we hope to move, every individual unemployed in the United States who registers either for a job at the U.S. Employment Service or at a private employment service, or for unemployment insurance would have information filed on the nature of his particular skill, educational background, work experience, age, and other social and economic characteristics of importance for job location. Such information would be coded in terms of a number, such as the Social Security number, so that in the event that such an individual was able to obtain a job for which he was qualified, that individual's availability for a job would be dropped from the storage system. In addition, all graduates of high schools, whether or not they were seeking jobs, would also be listed in the same manner on the assumption that they might possess skills for which jobs would be available in the United States.

On the employer side, all job vacancies would have to be reported to the U.S. Employment Service indicating the expected duration of the vacancy, the nature of the skills indicated as necessary to fill the vacancy, and other social an economic characteristics of the individual for whom the vacancy might be a possible source of work. In addition, all expected vacancies within a 12-month period would have to be listed by employers, thus providing an early-warning system of expected unemplo ment in a specific industry and community. This information would be gathered at each of the local U.S.E.S. offices in each of the states. Each state would then have a central collection agency which would then put all of this information, coded properly for identification purposes, on tapes from which the information would be brought together in a central skill-job-locator system. This locator system would quickly match jobs available—no matter where the job may be found in terms of geographical area and industry—with the skills which have been posted for each of the individuals seeking work. The matches which fall out of comparisons of skills and job openings would then provide a means for communication without delay to the concerned individuals. Communication would take place simultaneously, as a matter of fact, in order to permit both the employer and the prospective employee to make contact with each other. Depending on the scarcity of the skill and the proximity of the employer to the employee, there might be indicated the provision of mobility funds, either from industry or from the government. But in any case, at least there will have been achieved a matching of a job opening and the individual who is seeking that particular sort of job, or for which his skills provide him with a basis for doing the job. This second point is important because more and more we have begun to recognize the need for transfer from one particular occupational skill to another. In some cases, the skill

is such that we can move quite easily between industries; hence it's important that the information which is stored is of such a fundamental or generic nature that the language of the job description itself does not fall into folkways or traditional terms which might limit placement on the basis of a restrictive job description "cliché" rather than on the basis of skill descriptors which may qualify an individual for a number of jobs with completely different titles.

What I have discussed thus far would be an ambitious program to get under way immediately on a national level. However, a great deal can be done at the local level. By local level, I would begin within the confines of a major metropolitan area rather than on a larger regional basis. By starting at the local level, there are several assets. To begin with, the local U.S. employment services does, to some degree, although a limited one, know the local labor market. It does avail itself of a certain limited number of information inputs from employers with regard to contemplated changes in the labor force. It also, to some degree, makes use of what information is available on the nature of forthcoming graduates from vocational education programs and the nature of the skills which they may possess. Building on this rather primitive basis, we can begin to design at the local level a more formalized system of information acquisition, storage, retrieval and dissemination with respect to labor market structure and needs, both on the supply and demand sides. An additional asset, if we start at the local level, results from the fact that co-operation between employment services of the individual states varies rather widely and in most instances we find that the individual U.S. employment services within a state act on an autonomous basis. In some instances there is a bonus which results from the fact that some school systems have already begun to develop formal and standardized procedures for storing information on graduates. This is done with the objective of following up on graduates after their departure from the school system in order to determine what the nature of their work in the job world was after leaving school and the degree to which the training, especially vocational, affected their success in finding work as well as the nature of the work which they found. These information systems can be taken over and utilized for some of the initial information inputs which will be necessary on graduates as well as non-graduates coming from the school system and entering into the labor force.

In addition to the use of computer systems for providing a means of communication between the demand and supply sides of the labor market, there is also a fascinating opportunity to utilize the computers as a current analytical tool for determining the relationships between various economic, social, and educational characteristics of the employees or unemployed labor force in the population and the degree to which these characteristics affect mobility, job-seeking patterns, ease of placement, labor turnover, and other factors which are of prime consideration in the development of manpower policies. At the present time, we know very little concerning these sorts of factors and the degree to which there are correlations between these factors and other behavioral characteristics and placement possibilities of the unemployed as well as the employed in the labor force. By use of computers, we may also be in a far better position to determine the degree to which individuals with a broad array of skills may be underemployed in our economy. Once having established the basis for gathering information and communicating this information on individuals seeking jobs as well as employers seeking skilled individuals for jobs which are available, there is the potential for continuing to gather information which can be coded on the basis of Social Security numbers with regard to types of skill, hours of work, units of output or productivity, and other such economic and social characteristics while the individual is actually employed. This information can be collected, collated and analyzed on a current basis and, with such information being available, there is the possibility of our being able to move away from the rather limited sample which we now use to measure unemployment in the United States. Further, it presents us with the intriguing possibility of being able to put a tracer on individuals who have been displaced for various reasons from an industry, the degree to which they find new employment, the nature of the new employment which they find and the

levels of wages which these individuals are able to obtain in these new forms of employment. This is particularly intriguing because, at the present time, the major argument raging between economists concerning whether or not unemployment is due to structural factors, including technological unemployment, or too low a level of aggregate demand has been subjected to no truly rigorous research treatment. Each side of the argument obtains major support from deduction, logic, and scanty data rather than from large-scale analytical research procedures involving survey techniques and directly relevant primary data. Before closing, I would like to suggest one additional fascinating potential application of computer technology to a difficult manpower problem—that of designing more effective training systems. Among these systems I would include both vovational training at the high school level and adult re-training at the post high school level. If we regard this sort of training as a scheduling of a mix of sequential, overlapping and concurrent phases, where the optimal situation is one where we can forecast time "bottlenecks" and choose options which shorten or eliminate these bottlenecks, we are really considering the application of P E R T and critical path method techniques for training and curricula design problems. The scheduling of education, very much like the scheduling of a Polaris project, calls for the same awareness of the trade-off potential between time and money and the development of P E R T network or critical paths which permit the design of a system more concerned with producing an end product in less time rather than being concerned with a marginal savings of funds. This is not only of significance with respect to scientists and engineers but also with regard to potential juvenile delinquents and heads of families who, without work or meaningful job roles, represent serious personal and social costs with which we are already becoming familiar.

THE COMPUTER REVOLUTION AND THE SPIRIT OF MAN

Robert H. Davis
System Development Corporation
Falls Church, Virginia

Many of us are gradually becoming aware of the fact that an enormous number of Americans live in abject poverty. While estimates of their number vary, it is certain that millions of people in this country live in chronic need, perhaps as many as one out of every four of five citizens.[4] Children, the sick, and the aged constitute a large part of this number; but also included among them are approximately four million unemployed Americans on whom millions of additional citizens once depended for their support. Many of these people are victims of automation and the computer revolution. John Snyder, President of U.S. Industries, has estimated that automation is a major factor in displacing 40,000 workers per week. The Department of Labor, which is more conservative, estimates the rate of displacement through automation at 4,000 per week.

But poverty is not the only threat to those who have been displaced by automation. Even if the unemployed are provided an adequate standard of living, millions will still be threatened by psychological problems which have their roots in two conditions of the contemporary American scene. First, most unemployed people hold values based on the Protestant ethic—values which are ill-suited to a world in which there is not enough work to go around. And second, there is the machine itself, particularly the computer, which is presented to the worker as being faster, more accurate, more reliable, and in short, better than he is. In this essay I would like to focus not on the physical plight of these people but on the psychological problems they face.

Underlying many of the psychological problems which the unemployed face in trying to adjust to their condition is the fact that the prevailing value system in this society places great stress on the virtue of work. Although many observers have commented on this, one of the best known was Max Weber who called it the Protestant ethic.[9] The central notion in the Protestant ethic is the idea that labor is noble whereas idleness is immoral. The fact that Weber, a German, chose an American, Benjamin Franklin, as the exemplar of the Protestant ethic emphasizes the degree to which this country is uniquely identified with it.

The aphorisms of Franklin are rich in their reference to the moral value of a man's labor. Franklin, at one point in his career, set out to distill the essence of moral perfection into 13 virtues, one of which was *industry*.[3] "Lose no time," he advises. "Be always employed in something useful; cut off all unnecessary action."

In Poor Richard's Almanac, Franklin makes these additional observations on work and idleness:

> "At the working man's house hunger looks in, but dares not enter."
> "Plough deep, while sluggards sleep, and you shall have corn to sell and keep."
> "Trouble springs from idleness."

No twentieth century American is entirely free of his cultural ties to Benjamin Franklin or the Protestant ethic. Despite occasional jokes to the contrary, most of us feel uneasy when we are away from the daily working routine too long. Indeed, as our jobs become more abstract and it becomes more difficult to point to concrete products resulting from our most intensive labors, many of us are plagued with doubts. Are we in fact needed? Are we paying our way? Or are we perpetrating a gigantic hoax on a world too confused to recognize what it does, in fact, need?

Our uneasiness is rooted in the fact that work is the principal avenue by which Protestant man fulfills his potential as a unique and creative human being. Although our daily life is filled with numerous activities, work and sleep consume the largest part of our days; and in no other sphere is the potential for self-realization so great as work. To question my work is to threaten my value as an individual; to deprive me of work is to take from me the opportunity to give meaning to my life—the opportunity to achieve and to be recognized by my fellow human beings. The important point to note here is this: a man's labor does more than supply the necessities of life; in this culture, it also feeds hidden and only dimly understood psychic needs as well. Though a man may have more money than he requires to satisfy his own needs and those of his family, he still feels impelled to work. Work is an autonomous activity: it is good in and of itself, having a value surpassing the immediate ends it serves. "God gives all things to industry," says Poor Richard. To *not* work, on the other hand, is bad. "Be ashamed to catch yourself idle," as Poor Dick says.

Though it may be otherwise on some far off Pacific Island, in this culture the average man who does not work feels guilty and useless, and this, unfortunately, is true even though he is unable to work through no fault of his own—because he is too sick or too old or too young or too poorly trained.

While the unemployment statistics do not tell the whole story, they suggest that there is a great deal of psychological, as well as physical, misery in this country.[11] The plight of such "special" manpower groups as the young, the old, and the minorities is particularly tragic. Unemployment rates for teenagers are three times the national average and have risen almost 70% in the past five years. Older workers—those over 45 years old—are similarly disadvantaged, finding it difficult to find a new job once they lose their existing one. They account for a disproportionate percentage of the long-term unemployed, and many older workers "retire" prematurely to avoid the psychological stress of hunting for non-existent jobs.

Minority groups are hardest hit. Unemployment rates for non-whites are in almost all cases twice the rate for whites—regardless of educational level, age, sex or skill. This means that non-white unemployment rates for teenagers amount to about one potential worker out of three; and in some areas, only one youth out of two has found a job.

One could go on citing such statistics as these endlessly, but that is really unnecessary. These data point up the central fact that millions of people are caught in a fearful conflict. They are unable to find jobs in a culture where the prevailing ethic is "work or be damned."

These statistics appear to have had very little impact on the drive to replace men with machines. Indeed, the moral imperatives which guide the computer revolution seem to take for granted that man will somehow bungle through if the machine is properly tended. Writing in *Harper's* in 1951,[6] two Canadian physicists, E. W. Leaver and J. J. Brown, suggest that widespread automation will bring an era of peace and creative human development. Their basic principles, which will presumably bring about this new era of peace and prosperity, are notable for their emphasis on the machine, rather than on man himself, as the source of human salvation.

1. Machines should replace men wherever possible.
2. Men should not be used for routine operations if machines can do the work.
3. Automaticity of machines should be encouraged.
4. Automaticity of men should be discouraged.
5. Men must be ancillary to machines.

To a generation which has grappled with such earth-shaking problems as the control of nuclear weapons, the belief that machines should replace men or that men should be ancillary to machines may seem trivial. But in point of fact, it may be as revolutionary a change in the beliefs of man about himself as the shift which occurred with the widespread acceptance of the heliocentric rather than geocentric view of the universe.

Before the general acceptance of the heliocentric view of the solar system, most people saw a special significance in the fact that the sun and stars apparently circled the earth; it seemed to imply that man, who was obviously first on earth, must therefore enjoy a special relationship to the entire universe. But even then, men could still look around them and find in their own handiwork something special, something which set them apart from other living things. Veblen, in fact, considered a sense of workmanship chief among the instinctive dispositions of man.[7] The new doubts which the machine has cast on the capabilities of man are sufficient to shake his faith in himself to the very roots.

In the past, as machines extended human capabilities, they freed men from much of the drudgery that characterizes more primitive societies. Even today in many underdeveloped societies, virtually every member must devote all of his waking hours to the finding and preparation of food. There is not enough excess capacity in these societies to free more than a few individuals for other kinds of activities. Under such conditions, machines make good economic sense. While the rationale behind the quest for newer and better machines is still largely economic, we seldom ask what we are "freeing" men from and what we are "freeing" them for.

Underlying the relatively high value which we place on the machine is the notion that men really aren't very efficient mechanisms. Machines can do many things better than men, and some would assert that machines can do *most* things better than men. We are told that they are faster; that they are more accurate; that they are more dependable; and there are those who believe that computers rather than men should really make many decisions.

It would be well to note that it makes very little difference, for the purpose of this discussion, whether the machine is or is not superior to man. In many ways the issue is academic. *What really matters is the extent to which the average person is convinced that the machine will replace him because it is better than he is.* With the enormous prestige of science standing behind us, I submit that we have unwittingly convinced the average man (who may not in any case hold himself in very high esteem) that he is in fact not really worth very much. This, I fear, is a most serious mistake, first because of the impact it will ultimately have on our culture, and second, because it is not true!

Clearly the proposition that men should be replaced by machines because they are better than he is inconsistent with the Protestant ethic. If a man must work to feel worthy and needed and we persist in eliminating him, then the result must inevitably be a profound and possibly disastrous change in the fabric of our culture. While no one can predict precisely the social consequences of this state of affairs, most social scientists to whom I have talked are not optimistic.

To the sociologist, the disparity between the prevailing Protestant ethic and the realities of modern life as experienced by many unemployed people leads to an alienation between the individual and his reference group. Emile Durkheim[2] in his comprehensive study of suicide used the term "anomie" to describe the condition in which an individual is constantly forced to compromise his established values when they conflict with the existing reality—a condition which inevitably results in a feeling of separation and rootlessness.

From the viewpoint of the psychologist, most problems are rooted in conflict and frustration. When the society places inconsistent demands on the individual, insisting, for example, that he both work and accept the fact that jobs are not available, he struggles to resolve the inconsistency. In the absence of more effective response alternatives, the individual can be expected to invoke certain well established psychological mechanisms in order to adjust to continuing conflict and frustration. Aggression, regression, fixation, and in the face of prolonged

frustration, apathy, withdrawal and resignation, are mechanisms used to some extent by normal individuals. But when tension persists despite all efforts to remove it, and the individual resorts to inappropriate psychological mechanisms as a "way of life," the consequences are potentially very serious for his mental health.

The clash between the Protestant ethic and automation threatens to place literally tens of millions of Americans into what is essentially an irresolvable conflict situation. In the absence of socially acceptable alternatives, the most likely result of this is the relatively permanent adoption of one or more inappropriate psychological mechanisms. This is basically an unhealthy state of affairs.

Many advocates of the machine-over-man position declare that we will automatically adopt healthy alternatives—that we are entering a new era where the workman will turn to culture and the arts, thereby resolving the conflict in a socially acceptable and useful way. This seems unlikely.

According to statistics published by the Department of Labor,[11] the following educational levels can be expected to prevail in the 1960's:

For youths, if current trends continue, approximately 30 per cent of those entering the labor market (7½ million young people) will not complete high school. Approximately 2½ million of these will not even enter high school.

For non-white workers, the situation is even worse. As of March 1962, one out of every three non-white workers had not completed an elementary education, and only one in five had completed high school.

These educationally deprived groups, who will be the hard-core unemployed of tomorrow, do not appear to be particularly promising candidates to lead America in a new cultural renaissance—nor do the masses of older employed Americans who suddenly find themselves displaced by machines.

In Oklahoma City, in 1960, an intensive effort was made to help displaced workers after the closing of the Armour plant. One hundred and seventy people were tested, but only 60 showed promise for retraining. As a matter of fact, out of a total of 431 workers invited to be interviewed, only 143 men and 27 women completed both tests and interviews.[10]

In another recently published study,[13] the Labor Department compared the characteristics of all unemployed workers with those of 30,650 trainees enrolled in Federally financed retraining programs. Older workers are poorly represented in such programs. Only 5.7 per cent of trainees are over 45 years of age, and yet, 28 per cent of the unemployed fall into this age group. From the report, it is not clear why these people are so poorly represented, but one suspects that they are simply not prepared to build new lives for themselves through retraining. Whenever they are called upon to change their ways dramatically, older people must overcome a lifetime of learning and powerful habits built up through years of conditioning. The report notes that many older people are undereducated and unskilled and have little to offer the economy. It is reasonable to suppose that many of these people are undereducated because even in youth they were not highly motivated to learn, and that many of them simply have no wish to learn now. On the other hand, prodded by the Protestant ethic, they do have the desire to play a useful role in this society. It seems completely naive to believe, as some apparently do, that this need can be satisfied by social activities which demand a radical reorientation of their existing modes of behavior.

The new society will provide the potential for additional leisure, but a large proportion of the population is unfortunately ill-prepared for leisure. They are ill-prepared, first, because leisure is antithetical to the Protestant ethic, and second, because the enjoyment of leisure requires preparation for it. Relaxation may come naturally to us as children. But the ability withers under the regimentation of a civilized society, as evidenced by the emotional and psychological preparation required by most men who have worked all their lives and who are suddenly faced with the full-time leisure of retirement.

Will the unemployed voluntarily turn to education in search of a solution to their conflicts?

While I am not aware of any research directed at answering this specific question, a recent study [12] by the National Opinion Research Center provides some interesting information about the kinds of people currently participating in adult education programs. Although one out of five adults follows some plan for leisure-time education, the "typical" participant is apparently not one of the "other Americans." Indeed, the "typical" participant is described by the authors as a man or woman who has completed high school, has an above-average income, a full-time white collar job, and is a white married Protestant living in a city or suburb.

What very few people seem to fully understand is this: large numbers of unemployed people lack the most elementary academic skills, such as the ability to read well. In view of the fact that millions are insufficiently trained to find work, or even warrant retraining, and that many more are past their prime, fixed in their ways, and poorly adapted to change of any kind, it seems unlikely that they will turn to cultural interests.

Nor do these remarks apply solely to the laborer. It would be foolish to believe that the computer revolution will stop with the man in the denim shirt. On the contrary, white-collar automation is at least as great a threat as blue-collar automation. Today, white-collar workers being replaced are those who are engaged in basically routine tasks, but if the past is any guide to the future, the "clip level" will gradually rise to encompass literally millions of additional white-collar workers. In the crudest terms, computers are installed because they make economic sense, and they make the most economic sense where they eliminate high-cost manpower. Managers aren't installing computers for the fun of it; they are installing them because they save money, largely by doing tasks which would otherwise be performed by humans. To review each of the areas in which this is true would be both tedious and unnecessary. Airlines, banks, brokerage houses, manufacturing firms—virtually all sectors of the American economy—are discovering the simple fact that computers cut labor costs.

Whether white-collar workers will turn, en masse, to the classics, art, music, or even do-it-yourself projects, is not easy to foresee. But even if millions do turn to cultural pursuits, it is safe to predict that millions will not. What of them? Like their friends in denim shirts, they too are products of a Protestant ethic.

To summarize, millions of unemployed Americans, and millions more who may soon be displaced by automation, are psychologically threatened from two directions at once. First, there is the threat created by the fact that they hold a value system—the Protestant ethic—which is contradictory and ill-suited to the world in which they find themselves. And second, there is the threat presented by the machine itself—the computer—which makes the individual feel insignificant and inferior by comparison.*

What are we to do about the psychological problems I have described?

I have shown that the continuing process of automation challenges one of the most fundamental factors in the psychological makeup of the American individual. While we are probably unwilling and even unable to stop automation, it is within our power to change the nature

*Although I am convinced that what matters here is not whether machines are *really* superior to men but what people *believe* to be true, I cannot leave this topic without making one or two comments. Many authorities appear to feel that machines will some day simulate all of the important properties of human intelligence. As a psychologist, I am frankly a skeptic.

In a recent article published in Datamation,[1] Paul Armer deplores the fact that we keep redefining intelligence so that it is always just out of the reach of the machine. The fact of the matter is "intelligence" and "thinking" have always been concepts just out of the reach of scientific psychology, as well. We don't really know how to define them in an entirely satisfactory way. Indeed, *that is the problem.* I'm sure that *if* these terms are ever defined in a way which encompasses the full richness of the phenomenon to which they generally refer, then it will be possible to simulate them.

But, to be satisfactory, the definition will have to include some extremely frustrating aspects of human behavior. For example, any really adequate definition of thinking will have to recognize the fact that perceptions are selective and shaped by past experience; that the mind wanders down strange and wonderful unprogrammed paths; and that the most creative thoughts are sometimes elicited by the most illogical and bizarre associations. In short, the very features of man which make him slow and unreliable may be his most valuable properties.

of automation by making it more sensitive to human variables. To the extent that our analyses of the benefits of automation ignore human costs, they are incomplete. There are costs associated with retraining, with the support of the unemployed, with social dislocation, and with the psychological misery of the displaced worker. In general, when the benefits of automation are calculated, these costs are ignored. Taxation may spread such costs over a broad base, but they are still enormous—and they will continue to grow as automation expands. It is important that computer scientists become sensitive to these human and social costs so that systems analyses include *all* of the relevant variables.

Second, although the task will take decades, we must begin to change the Protestant ethic. Changing the Protestant ethic will involve a dramatic reorientation of our society. Most important, we will have to learn to live with the fact that the day is not far off when computers and improved machines will make it impossible for ever-increasing numbers of Americans to fulfill their potential through work. Many of our children can expect to spend a large part of their lives in leisure. To this end, public education should modify its curricula to prepare our children and grandchildren not only for work but for the more profitable use of free time as well. Unlike many who see in every subject which is not blessed by Admiral Rickover a direct threat to the American way of life, I believe we should begin to emphasize in our schools the constructive use of leisure by encouraging our young people to participate in the full range of life's activities, including art, music, drama and similar fields which today are seen by many to be frills. Also, the society should recognize and reward excellence in these fringe areas as much as it does in more traditional fields. While I do not believe we should turn our backs on science, we should not allow it to consume us either.

Third, we must re-examine the economic base of our society. Up to the present time, we have forced man to work if he wished to receive an income. If we can no longer provide work in the traditional sense for everybody, we must reconsider how the unemployed are to be provided with resources and what volume of resources they should receive. Already, some writers have suggested that every individual will have to be given an absolute right to an income adequate to live his life in dignity.[8]

Fourth, there is the requirement that we educate every citizen to the limit of his abilities. To the extent that people are needed at all in the emerging, new society, they must be educated. Most of those who are not educated will be cast ruthlessly aside. Furthermore, education is a profitable way to consume time which might otherwise be spent in antisocial or socially maladaptive ways. Therefore, the right to an education is an essential guarantee—not only for the protection of the individual but for the protection of society as well.

Finally, we need more information about the problems of the unemployed. We are in need of better psychological data about the displaced worker, his attitudes toward society, the way he spends his time, and his fears for the future. We must try to determine the extent to which the conflicts I have postulated lead to an alienation of the individual from society and how we can prevent this from happening. We need to know more about how to prepare people to accept change, particularly older workers. We must refine our tools for predicting future trends so as to properly prepare our young people for the world they will someday face.

Carl Jung[5] once remarked that in science "the individual man and, indeed, all individual events whatsoever suffer a leveling down and a process of blurring distorts the picture of reality into a conceptual average." In the scientist's search for lawfulness, he abstracts away complexity and reduces the individual to a statistic. So it is with all of the statistics which I have cited. The individual is lost. But if we look beyond the abstraction to men, as living, breathing individuals, there is much to be concerned about in this affluent society of ours.

REFERENCES

1. ARMER, PAUL. "Attitudes Toward Intelligent Machines," *Datamation*. March 1963, pp. 34-38.
2. DURKHEIM, E. *Suicide A study in Sociology*. Glencoe, Illinois: The Free Press, 1951.

3. FRANKLIN, BENJAMIN. *The Autobiography of Benjamin Franklin.* New York: Books, Inc., n.d.
4. HARRINGTON, MICHAEL. *The Other America.* New York: W. W. Norton & Company, 1962.
5. JUNG, C. G. *The Undiscovered Self.* New York: The New American Library of World Literature, Inc., 1961.
6. LEAVER, E. W., and BROWN, J. J. "Electronics and Human Beings," *Harper's Magazine.* August 1951, pp. 88-93.
7. LERNER, MAX (Ed.). The Portable Vebelen. New York: The Viking Press, 1958.
8. THEOBALD, ROBERT. *Free Men and Free Markets.* New York: Clarkson Potter, 1963.
9. WEBER, MAX. *The Protestant Ethic and the Spirit of Capitalism.* New York: Charles Scribner's Sons, 1958.
10. "Automation: Report of the Armour Committee." *Bureau of National Affairs,* No. 419, June 23, 1961, 16:621.
11. "Nation's Manpower Revolution." *Hearings before the Subcommittee on Employment and Manpower of the Committee on Labor and Public Welfare, United States Senate, Eighty-Eighth Congress, First Session, Part 4.* Washington: U.S. Government Printing Office, 1963.
12. *The New York Times.* May 20, 1963, p. 31.
13. *The New York Times.* December 1, 1963, pp. 1 & 48.

NEW DIFFERENCE EQUATION TECHNIQUE FOR SOLVING NONLINEAR DIFFERENTIAL EQUATIONS

James M. Hurt
Associate Engineer
IBM Corporation
Data Processing Division
Kingston, New York

INTRODUCTION

This paper will discuss a new mathematical technique for the solution of nonlinear differential equations. The types of equations and nonlinearities presented are those associated with feedback control systems. Originally developed as a method for simulating control systems, the technique was verified during a recent study of complex aircraft design simulations. It is now being used in a man-in-the-loop Gemini spacecraft simulation.

Since these design simulations include man as a component part, it is imperative that they be performed in real time. For many years, the analog computer with its parallel computing ability has been used for the real-time design simulation of aircraft. In more recent years, the analog computer has assumed the same role for missiles and space vehicles. Presently, aircraft systems have become more complex, space vehicle missions require more hours of simulation time to complete a maneuver, and high-performance missiles require more logical decisions in their phases of flight. Because of these conditions, it has become more difficult to expand, use and maintain the large analog systems which are required to perform real-time simulation.

Until the present technique was developed, a general purpose digital computer was unable to achieve a real-time solution for the more complex design simulations. The primary obstacle to the digital approach was the excessive computing time required for simulating the vehicle's control system and rotational motion equations by available numerical techniques. The roll, yaw and pitch channels of the aircraft used as an example are represented by fifteen linear and nonlinear differential equations in addition to algebraic equations. The highest input frequency is approximately three cycles per second; the maximum transient response frequency is 60 cycles per second. The solution time for these equations on the IBM 7090 by Runge-Kutta integration is 3.0 seconds for each second of flight time. When solved by the new technique, the same equations require only 0.17 seconds of computation time for each second of flight time. This solution-time reduction, in addition to a reduction in the computation time for the vehicle dynamics equations, permits a digital computer to perform the simulation in real time.

This paper will describe the theory of the new technique as applied to the flight simulation problem. Autopilot and power controls equations, and vehicle dynamics equations will be used as examples.

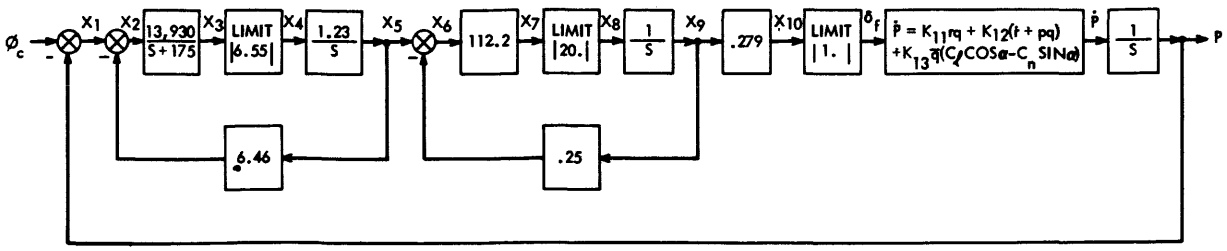


Figure 1. Transfer Function Diagram for Aircraft Roll Channel.

PROBLEM DEFINITION

Figure 1 illustrates the Laplace transfer function diagram of an aircraft roll channel.* The yaw and pitch channels are of similar form and will not be discussed. This transfer function diagram represents the mathematics of the vehicle's roll rate as a function of autopilot (ϕ_c) input. Additional inputs are applied at point X_5. The first two inner loops contain nonlinearities in the form of limiting devices. These devices limit the magnitudes of the variables X_4 and X_8 to 6.55 and 20.0, respectively. The first inner feedback loop is a high-frequency control unit which accepts autopilot signals as input. Its internal frequency response is up to 60 CPS and drives the second inner feedback function. This second feedback transfer function is a low-frequency unit and is the mathematical representation of the hydraulic unit of the roll channel.

The next to last block gives the differential equation for the roll acceleration. The equation contains continuous nonlinearities in the form of the time-varying coefficients C_l, C_n and α, and the crosscoupling terms, r and q. The time-varying coefficients are functions of altitude, speed, angle of attack, etc., while the crosscoupling terms introduce the effects of yaw and pitch on the roll of the aircraft.

The problem of concern is the solution of the nonlinear differential equations for the roll channel (yaw and pitch also) at a solution rate which will give results that are accurate for a design analysis, and yet permit real-time operation of the entire simulation.

*Laplace transfer functions rather than differential equations will be used to specify the mathematical models. The use of the new technique to be discussed is greatly aided by starting with transfer functions.

SOLUTION METHOD

The nonlinear differential equations are solved by an appropriate set of difference equations. These difference equations are derived by first computing a z transform† of the Laplace transfer diagram and then reducing the transforms to simple difference equations. The z transforms are used in a manner which differs considerably from their normal usage. By properly computing z transforms for the linear portions of the transfer diagram, and then combining the linear portions as though they were separated by samplers, it is possible to arrive at a transformation which, when reduced to difference equations, will permit an accurate and rapid solution of the nonlinear system.

Figure 2 is a feedback control system with a nonlinearity (the limiter) in the forward loop. The limiter prevents the magnitude of the variable $X_4(S)$ from exceeding the limit K_L. The limiter behaves as a variable gain element. For the nonlimiting mode, the gain K is unity and, when limiting occurs, the gain is such that: $X_3(S)K = K_L = X_4(S)$. When the system is operating in the nonlimiting or linear region, the forward loop transfer function is $G_1(S)G_2(S)$ and the system transfer function is

$$\frac{X_5(S)}{X_1(S)} = \frac{G_1(S)G_2(S)}{1 + G_1(S)G_2(S)H(S)}.$$

For this nonlimiting mode, a z transform of the total transfer would be computed as $Z\left[\frac{G_1(S)G_2(S)}{1 + G_1(S)G_2(S)H(S)}\right]^{\ddagger}$. This transfer

†z is the sampled data variable and is equal to e^{ST} where S is the complex Laplace variable and T is the sampling period. For further information, refer to sources listed in the Bibliography.

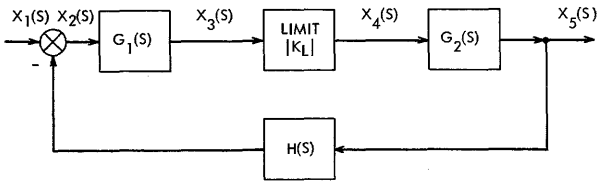


Figure 2. Feedback Control System with Limiting.

function has the correct roots and steady-state gain for a single impulse input. For any other input, the steady-state gain would need adjustment. This would necessitate adjusting the numerator polynomial coefficients. The denominator polynomial would be correct and would not need adjustment.

For the limiting case, the forward loop transfer function would no longer be $G_1(S)G_2(S)$, but $KG_1(S)G_2(S)$ where $K < |1|$. The limiter acts as a variable gain element. When $X_3(S)$ becomes larger than some predetermined value, limiting occurs such that $|X_4(S)| = K_L$. Thus, for a particular sampling instant and limiting gain K_1, the system is considered linear and the transfer function becomes $\frac{X_5(S)}{X_1(S)} = \frac{K_1G_1(S)G_2(S)}{1 + K_1G_1(S)G_2(S)H(S)}$. The resulting z transform would be $Z\left[\frac{K_1G_1(S)G_2(S)}{1 + K_1G(S)G(S)H(S)}\right]$.

Thus, for both the limiting and nonlimiting cases, the system at any sampling instant may be considered linear. This leads to the concept of a piece-wise linear system. The main problem which remains is to determine when the limiting occurs, and it is here that a departure from standard z transform techniques is made.

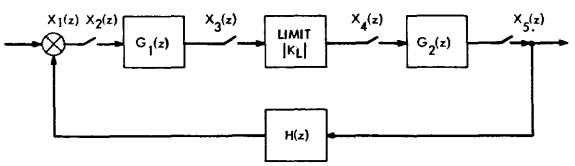


Figure 3. Sampled Data Feedback Control System.

‡This equation is written in operator notation. The capital Z indicates the z transformation operation. Thus $f(z) = Z[f(S)]$.

Some provision for stopping and checking for the limiting case must be made. This can be done by considering each element of the system separated by a sampler as shown in Figure 3.

In this form, the calculations proceed around the loop, one transfer function at a time. On each pass through the forward loop, the test for the limit is made before proceeding with the calculations. In this form, the system transfer function for the nonlimiting case becomes $\frac{X_5(z)}{X_1(z)} = \frac{G_1(z)G_2(z)}{1 + G_1(z)G_2(z)H(z)}$, which does not correspond with the nonlimiting transfer function $Z\left[\frac{G_1(S)G_2(S)}{1 + G_1(S)G_2(S)H(S)}\right]$ calculated previously from the Laplace system. The roots and gain of the transfer function will not agree with those of the z transform calculated from the Laplace transfer function. However, by appropriate gain adjustments in the forward and feedback loops, it is possible to adjust the z transform so that the roots and steady-state gain match. The difference equations obtained from this corrected z transform will give solutions which are within the accuracy required for design simulations.

Figure 4 shows the transfer function diagram of a feedback control network which will be used as an example to illustrate the technique.

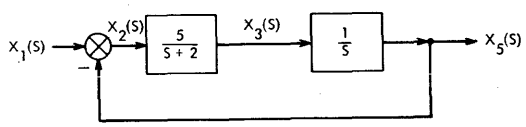


Figure 4. Linear Feedback Control System.

Since the system of Figure 4 is linear, an exact difference equation may be derived. The method consists of multiplying the Laplace transform of the input by the transfer function to obtain the output transform. The output transform is transformed into z transform notation. By dividing this output transform by the z transform of the input, the z transform transfer function is determined. This transfer function is then used to compute a difference equation which will give an exact

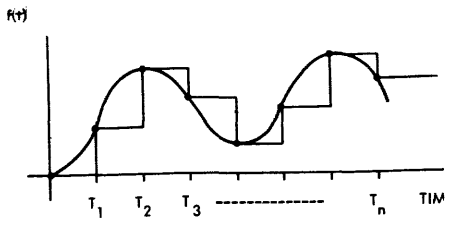


Figure 5. Arbitrary Input Sampled by Zero-Order Hold.

solution for the particular input used. However, for real-time design simulations the inputs are, in general, unknown and this method cannot be used for computing the difference equations. This difficulty is resolved by computing z transforms for step inputs or pulses of finite width, where the width of the pulse is equal to the sampling period. Any arbitrary input may be approximated by a stair-step input.

Figure 5 illustrates a function which has been broken up into steps.

This type of sampled input is obtained by a zero-order hold, and introduces, on the average, one-half of a sample period lag in the input. This lag is compensated by a lead which is characteristic of the z transform method. The transfer function of Figure 4 is

$$\frac{X_5(S)}{X_1(S)} = \frac{G(S)}{1 + G(S)H(S)} = \frac{5}{S^2 + 2S + 5} \quad (1)$$

The output transform for a unit step input is calculated by multiplying equation (1) by the unit step transform $\frac{1}{S}$.

$$X_5(S) = \frac{X_5(S)}{X_1(S)} \times \frac{1}{S} = \frac{5}{S(S^2 + 2S + 5)} \quad (2)$$

Equation (2) is expanded by partial fractions so that standard z transform tables may be used.

$$X_5(S) = \frac{5}{S(S^2 + 2S + 5)} = \frac{1}{S} - \frac{S + 1}{S^2 + 2S + 5} - \frac{1}{S^2 + 2S + 5} \quad (3)$$

Equation (3) may now be transformed into z notation through the use of transform tables.§ The sampling period used is T = 0.05 seconds. This rate is much higher than needed to produce satisfactory results. The choice of a solution rate is governed by the frequencies of the input function and transient response. The transient response frequencies are calculated from the roots of the system transfer function. To reproduce the response at this frequency, a sampling rate greater than but not less than twice the frequency should be used. Since an arbitrary input function is sampled in a stair-step manner, a sampling rate must be chosen to minimize the error introduced by the stair-step approximation. This error may be made insignificant merely by increasing the solution rate. This error is not present for step inputs since the difference equations are derived for step inputs. Thus, for step inputs, the solutions are exact at the sampling instant, regardless of the solution rate. A general criterion for choosing a solution rate is to pick one which will give satisfactory simulation results. The rate is much less than required for standard numerical techniques.

$$X_5(z) = \frac{z}{z-1} - \frac{z^2 - e^{-.05}\cos(.1)z}{z^2 - 2e^{-.05}\cos(.1)z + e^{-.1}} - \frac{.5e^{-.05}\sin(.1)z}{z^2 - 2e^{-.05}\cos(.1)z + e^{-.1}} \quad (4)$$

$$X_5(s) = \frac{z}{z-1} - \frac{z^2 - .94648z}{z^2 - 1.89295z + .904837} - \frac{.04748z}{z^2 - 1.89295z + .904837} \quad (5)$$

$$X_5(z) = \frac{z-1}{z} - \frac{z^2 - .8990z}{z^2 - 1.89295 + .904837} \quad (6)$$

The system transfer function in z notation is computed by dividing equation (6), the output response, by $\frac{z}{z-1}$, which is the step input in z transform notation.

$$\frac{X_5(z)}{X_1(z)} = \frac{X_5(z)}{\frac{z}{z-1}} = 1 - \frac{z^2 - 1.8990z + .8990}{z^2 - 1.89295z + .904837} \quad (7)$$

$$\frac{X_5(z)}{X_1(z)} = \frac{.00605z + .005837}{z^2 - 1.89295z + .904837} \quad (8)$$

The difference equation is computed by cross

§ Refer to appendix for a list of basic z transforms.

multiplying both sides of equation (8) and solving for $X_5(z)$.

$$z^2X_5(z) - 1.89295zX_5(z) + .904837X_5(z) = .00605zX_1(z) + .005837X_1(z) \quad (9)$$

$$X_5(z) = .00605\frac{X_1(z)}{z} + .005837\frac{X_1(z)}{z^2} + 1.89295\frac{X_5(z)}{z} - .904837\frac{X_5(z)}{z^2} \quad (10)$$

In z transform theory, $\frac{1}{z}$ represents a time delay of one sampling period; $\frac{1}{z^2}$ represents a delay of two sampling periods; and, in general, $\frac{1}{z^n}$ is a time delay of n sampling periods, or nT. The inverse transformation of equation (10) back to the time domain thus becomes:

$$X_5(nT) = .00605X_1(nT-T) + .005837X_1(nT-2T) + 1.89295X_5(nT-T) - .904837X_5(nT-2T) \quad (11)$$

Equation (11) gives an exact solution at the sampling instants, nT, for the output of Figure 4 for step inputs (or for finite-width pulse inputs of width equal to the sampling period).

A new method of calculating the difference equations will now be discussed. The new technique is required for the case where a limiter is present. The same system of Figure 4 will be used. Normally, a limiter woud be present between the two elements in the forward loop of Figure 4. However, for the initial analysis, the nonlimiting mode is used and the limiting element can be replaced by an element of unity gain. The method, as stated before, is to compute z transforms for the linear portions, combine these portions for the overall transfer function, and then make appropriate gain adjustments such that the steady-state gain and roots agree with the linear gain and roots. The transforms for the linear portions are calculated directly from the transform tables with one modification. The transforms in the tables are transforms for a single-impulse input. Here, step inputs are used which, through the sampling process, become a train of impulses. The transforms for the single-impulse input do not give a correct value of steady-state gain for the impulse train. This difficulty can be resolved by using impulses of area T, the sampling period, or by multiplying the transform by T and using the unit impulse. The multiplication by T is a first approximation. The numerator will generally require further adjustment. For an integrator, the numerator is multiplied by T and no further adjustment is required. The integrator adjustment may be explained by reducing the sampling period. The z transform is a discrete process which approaches the Laplace as the sampling period becomes smaller. Consider the Laplace transform for integration $\frac{1}{S}$. From the tables, the corresponding z transform is $\frac{z}{z-1}$ which is multiplied by T to get $\frac{Tz}{z-1}$. The sampling period is reduced.

$$\lim_{T\to 0}\frac{Tz}{z-1} = \lim_{T\to 0}\frac{Te^{ST}}{e^{ST}-1} \quad (12)$$

$$\lim_{T\to 0}\frac{Tz}{z-1} = \frac{T(1+ST+\frac{S^2T^2}{2}+\ldots)}{(1+ST+\frac{S^2T^2}{2}+\ldots)-1} \quad (13)$$

$$\lim_{T\to 0}\frac{Tz}{z-1} = \frac{T+ST^2+1/2S^2T^3+\ldots.}{ST+1/2S^2T^2+\ldots.} = \frac{1}{S} \quad (14)$$

As the sampling period is reduced to zero, the z transform for integration becomes $\frac{1}{S}$, which is the Laplace integration transform as expected. However, if the T had been omitted, the limit would have been infinity which is incorrect. The z transforms for the linear portions of Figure 4 are calculated from tables.

The calculations for the numerator coefficient of equation (15) may be omitted and a constant K inserted. This coefficient will undergo a gain adjustment when the roots are corrected.

$$\frac{5}{S+2} \to \frac{T5z}{z-e^{-.1}} = \frac{.25z}{z-.904837} = \frac{Kz}{z-.904837} \quad (15)$$

$$\frac{1}{S} \to \frac{Tz}{z-1} = \frac{.05z}{z-1} \quad (16)$$

The gain adjustment could have been made in the integrator coefficient with the same output

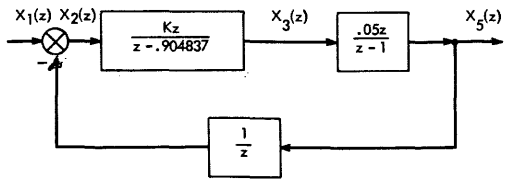


Figure 6. Sampled Data Feedback Control System—Loop Gain Unadjusted.

results. The only effect would be a time shift in the response within the loop. The limiter would limit at a slightly different time. Figure 6 shows the transfer function at this point. A delay of one sample period has been inserted in the feedback loop. Since the numerator and denominator of the forward loop transfer function are the same degree, the corresponding difference equation relates the present value of $X_5(z)$ to the present value of $X_2(z)$. However, $X_2(z)$ is equal to $X_1(z)$ minus $X_5(z)$. Therefore, without a delay in the feedback, the computer would be faced with the problem of calculating a number which depends on itself. This problem is avoided by inserting a delay of one sampling period in the feedback loop. With the delay added, past values of $X_5(z)$ are used to calculate the present value of $X_5(z)$.

The transfer function of Figure 6 is

$$\frac{X_5(z)}{X_1(z)} = \frac{G(z)}{1+G(z)H(z)} = \frac{K.05z^2}{z^2+(.05K-1.904837)z+.904837} \quad (17)$$

The coefficient for z in the denominator polynomial of equation (17) contains the unknown gain constant K. This constant originated from equation (15) where the z transform of $\frac{5}{S+2}$ was calculated. The reason given for keeping the numerator of equation (15) a constant K was that this numerator coefficient would undergo an adjustment when the roots were corrected. The denominator roots of equation (8) are the correct roots for the system of Figure 4. Since the roots determine the transient response of a system, the denominator of equation (17) must match the denominator of equation (8) if the system of Figure 6 is to perform as the original system of Figure 4. The denominator of equation (17) is matched to the denominator of equation (8) by solving for the gain constant K.

$$.05K - 1.904837 = -1.89295 \quad (18)$$

$$.05K = .011887 \quad (19)$$

$$K = .23774 \quad (20)$$

$$\frac{X_5(z)}{X_1(z)} = \frac{.011887z^2}{z^2-1.89295z+.904837} \quad (21)$$

Equation (21) is the final transfer function. It has the same denominator as equation (8) and the numerator coefficient is equal to the sum of the numerator coefficients of equation (8). Since the sum of the numerator coefficients of equation (8) equals the numerator

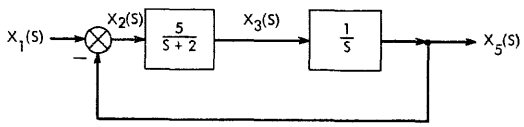


Figure 7A. Linear Feedback Control System.

coefficient of equation (21), the steady-state gain of (21) is equal to (8) as desired. The numerator is seen to be of an order higher than equation (8). This introduces a lead in the system output.

Figure 7 shows a comparison of the output response of the system of Figure 4 to a pulse

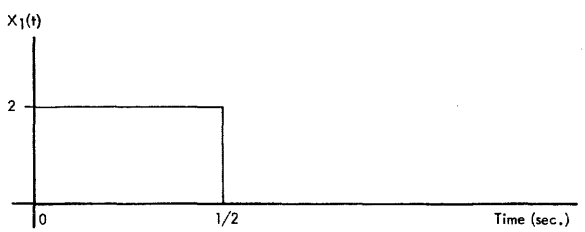


Figure 7B. Input Pulse.

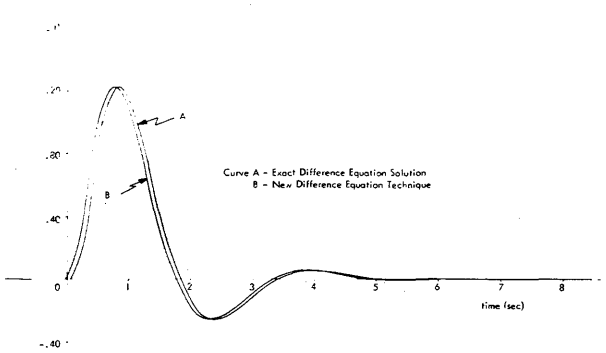


Figure 7C. Output Pulse.

input for the two methods of solution. Part A of the diagram shows the system; Part B represents the pulse input; and Part C gives the output responses. The results are for the nonlimiting mode. The solution which uses standard z transform techniques agrees exactly with the continuous solution given by equation (22). Equation (22) was found from the inverse transformation of equation (2). The input notation $X_1(t)$ has been changed to unit step notation $U(t)$.

$$X_5(t) = 2U(t) - 2e^{-t}\cos(2t) - e^{-t}\sin(2t) - U(t-1/2)\left[2 - 2e^{-t+1/2}\cos(2t-1) - e^{-t+1/2}\sin(2t-1)\right] \quad (22)$$

The solution as calculated from the difference equations derived from Figure 6 leads the exact solution, as expected, since a pulse input was used and no delay in step or pulse inputs is introduced by the sampling process. This lead may be programmed out, or may be desirable in cases where instruments are driven in a simulation. For an arbitrary input, the stair-step approximation introduces a delay in the input which cancels all or much of the lead due to the solution technique.

Figure 8A is Figure 4 with the limiter added to the forward loop. The limiter limits the peak magnitude of the variable $X_4(S)$ to unity. Figure 8B is the z transform of Figure 8A. It is the same as Figure 6 except for the addition of the limiter and the value for K. Curve B of Figure 9 is the difference equation solution of the system of Figure 8B for the pulse input

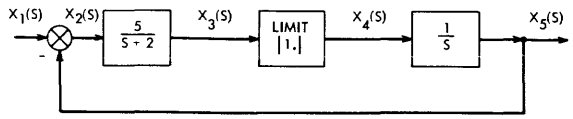


Figure 8A. Laplace Feedback Control System with Limiter.

shown in Figure 7B. Curve A is the solution of Figure 8A for the same input pulse. This solution was achieved by Runge-Kutta integration at a solution rate of 2,000 solutions per second. The difference equation solution leads

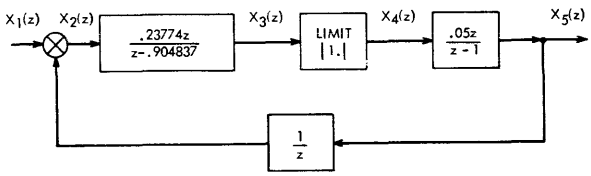


Figure 8B. Z Transform Feedback Control System with Limiter.

the Runge-Kutta solution but has the same amplitude and frequency. The difference equations were written from the linear portions of Figure 6 such that the computer would calcu-

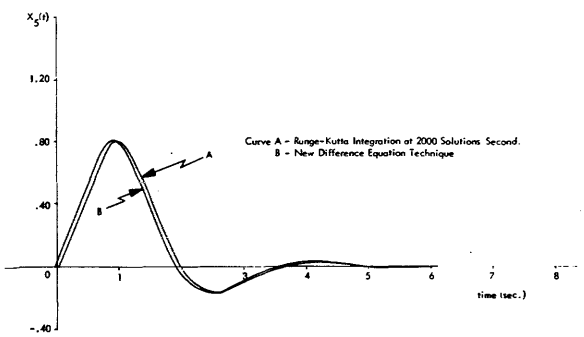


Figure 9. Output Response for a Pulse Input for Circuit Shown in Figure 8.

late around the loop a step at a time. This step-wise solution was necessary to check for the limit. The equations used were derived from Figure 8B.

$$X_2(nT) = X_1(nT)—X_5(nT—T) \quad (23)$$

$$X_3(nT) = .23774X_2(nT) + .904837X_3(nT—T) \quad (24)$$

$$X_4(nT) = \text{LIMIT}_{|1|} X_3(nT) \quad (25)$$

$$X_5(nT) = .05X_4(nT)+X_5(nT—T) \quad (26)$$

Figure 10 shows, in chart form, the effect of limiting on the system roots. A comparison is made between the root loci of the continuous system and the z transform system for several values of limiter gain. There are two z transform root loci; one is for a sampling period of T = 0.05, and the other is for a much slower rate of T = 0.2. The two z transform root loci correspond very closely to the Laplace for values of limiting gain near unity. This is true because the z transform equations were matched for the nonlimiting case K = 1. For smaller values of limiter gain, the roots of the z transform deviate slightly from the Laplace roots. This produces a small shift in the transient response frequencies.

Limiting Gain	Root: $S = R_e \pm jI_m$	
	Laplace and z Transform, T = .05	z Transform T = .2
1.000	-1.0 ± j 2.00	-1.0 ± j 2.000
.922	-1.0 ± j 1.90	-1.0 ± j 1.899
.848	-1.0 ± j 1.80	-1.0 ± j 1.797
.778	-1.0 ± j 1.70	-1.0 ± j 1.696
.712	-1.0 ± j 1.60	-1.0 ± j 1.595
.650	-1.0 ± j 1.50	-1.0 ± j 1.493
.592	-1.0 ± j 1.40	-1.0 ± j 1.393
.538	-1.0 ± j 1.30	-1.0 ± j 1.292
.488	-1.0 ± j 1.20	-1.0 ± j 1.191
.442	-1.0 ± j 1.10	-1.0 ± j 1.091

Figure 10. Root Locus Comparison.

ROLL CHANNEL EXAMPLE

This example will explain the important features of the z transformation of the roll channel of Figure 1. Figure 1 is expressed in Laplace notation, except for the roll acceleration ($\dot{p}$) equation. Before the z transform is computed, this equation is placed in Laplace notation. The equation for $\dot{p}$ contains time-varying coefficients and crosscoupling terms from the yaw and pitch channels. These crosscoupling terms have been expressed as a function of time only; however, they are mutually dependent functions. Thus, rolling of an aircraft will affect its yaw and pitch rates. Because of this interdependence of the crosscoupling terms, the equations for roll acceleration ($\dot{p}$), yaw acceleration ($\dot{r}$), and pitch acceleration ($\dot{q}$) form a system of nonlinear differential equations. For analysis purposes, each of these equations may be analyzed separately for a predetermined region of flight by simply considering the crosscoupling terms as additional inputs to the transfer function.

$$\dot{p}(t) = K_{11}(t)\dot{r}(t)q(t)\ [r(t) + p(t)q(t)] + K_{13}(t)q\ [\bar{q}_1 \text{ Cos } \alpha — C_n \$ \text{in } \alpha] \quad (27)$$

Since the transfer function for each equation is matched to give correct transient response and steady-state gain, the total system of equations will behave properly because the crosscoupling terms are treated as additional inputs to each channel. Equation (28) is the result of simplifying equation (27) by choosing an optimum region of flight.

$$\dot{p} = 65.2\partial_f + K_{11}rq + K_{12}\dot{r} + 19\partial_r — 12.2p \quad (28)$$

The flight region was chosen near the high performance end. This choice was made so that the coefficients in equation (28) would be near these maximum values. This enables the analysis of the root locus for the overall feedback system to be made at maximum loop gain. For a lesser performance region, the coefficients would be reduced. This would cause a reduction in loop gain and a shift of the operating point of the root locus. As previously shown, a good match between the actual system root loci and

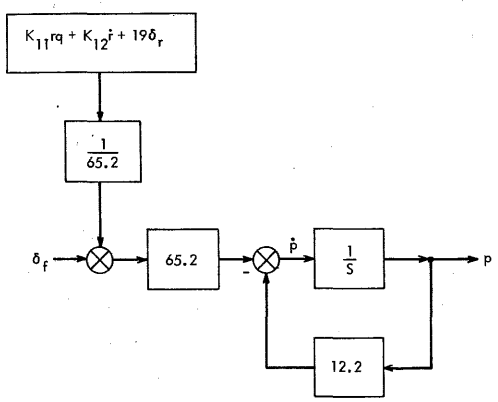


Figure 11. Transfer Function Diagram for Aircraft Roll Rate.

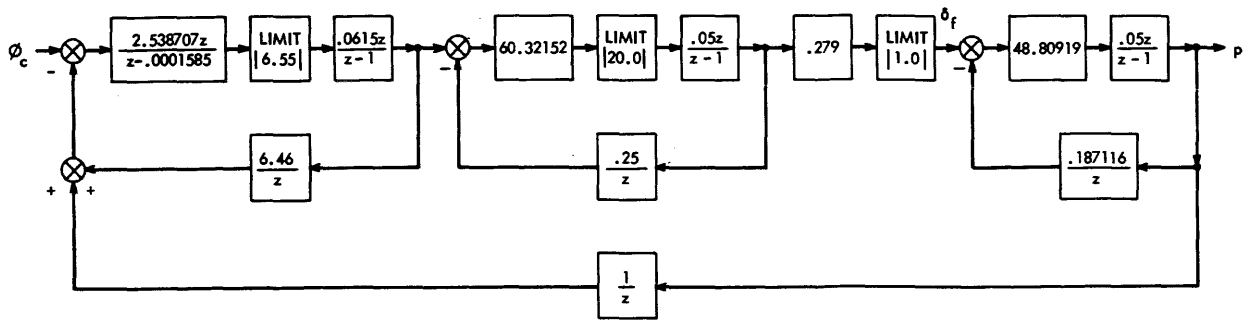


Figure 12. Roll Channel Z Transform Transfer Function (Roots and Steady-State Gain Unadjusted).

the z transform root loci is possible for lower loop gains.

Figure 11 is the transfer function diagram of equation (28). The diagram shows the cross-coupling terms as inputs.

The transfer function of Figure 11 is used in place of the portion from δ_f to p in Figure 1. Once this substitution has been made, the roll channel transfer function is transformed into z notation. This transformation of the forward loop proceeds one inner feedback loop transfer function at a time. The procedure for transforming each inner loop is the same as discussed previously and will not be repeated. A solution rate of 20 solutions per second was chosen. This rate gives satisfactory results and permits a real-time solution. Figure 12 is the resulting z transformation.

After performing the z transformation of the individual loops of the system, there still remains one final loop gain adjustment. This adjustment is necessary to match the roots and gain of the z transform system to the Laplace system. The adjustment in loop gain is determined by comparing the root locus of the z transform with the root locus of the Laplace transfer function. A root locus program‖ was used which computes points on the root locus for various values of gain (K) which the program inserts in the loop. The gain adjustment for the z transform system is determined by finding the roots of the Laplace system for K = 1, and then comparing the gain of the z system at the same point in the S plane.

Figure 13 is a plot of the two root loci in the S plane.

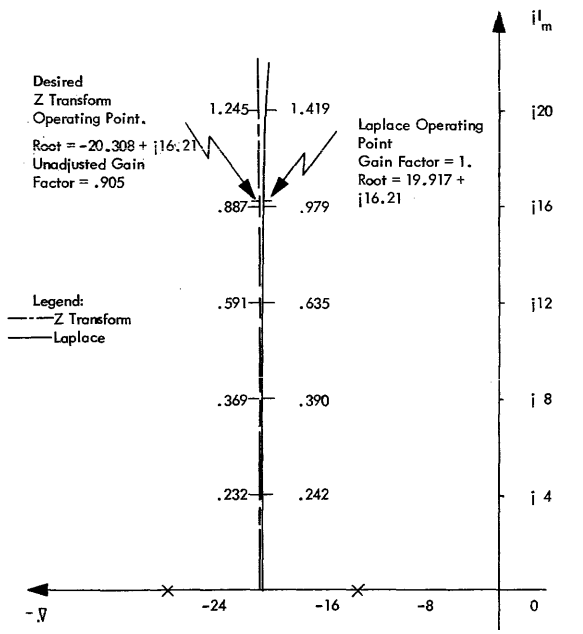


Figure 13. S-Plane Root Locus Plot with Gain Factor (K) as a Parameter.

Since the computer did not calculate roots for K = 1, a linear interpolation was used to determine the K = 1 point. The resulting root is S = —19.917 + j16.210. The adjacent z root

‖ The Control System Analysis Program computes root loci for linear, continuous or sampled data systems in open-loop form. The input data may be in either S or z transform notation, or any combination of these, and may contain transportation lags. Options include z transform computation pole-zero loci, and gain loci for a specified gain.

The method of computing the root locus is described in the paper "Numerical Methods for the Synthesis of Linear Control Systems" by Maurice E. Fowler. The paper was published in the 1963 Volume 1 of Automatica, pp. 207-225.

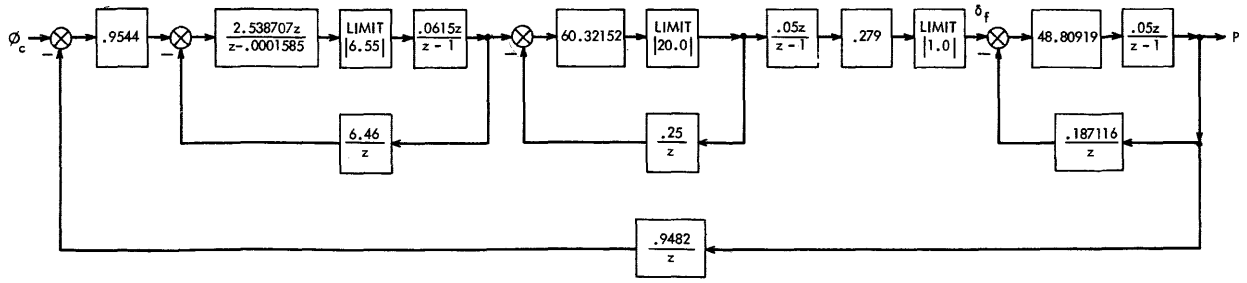


Figure 14. Roll Channel Z Transform Transfer Function (Roots and Steady-State Gain Adjusted).

locus does not have the same real part due to the characteristic lead introduced by the method; however, a close match may be obtained by using the imaginary part of the root. For $I_m S = 16.210$ on the z locus, the gain is determined by linear interpolation to be $K = 0.905$. Since this gain should have been unity, the z root locus was matched to the Laplace by reducing the z loop gain by a factor of .905.

Figure 14 is the final z transform of the roll channel. The loop gain adjustment was made in the forward and feedback loops. This was done to adjust the steady-state gain of ϕ_c to p, and also to obtain the correct system roots.

Figure 15 shows in chart form, the Laplace and adjusted z transform root loci. Values of gain less than unity may be thought of as a reduction in loop gain due to limiting in the forward loop. The values of gain shown in the chart are not the actual loop gains, but are multiplicative gain factors inserted in the loop.

Laplace			z Transform T = .05	
Root $S=R_e \pm jI_m$			Root $S=R_e \pm jI_m$	
R_e	I_m	Limiting Gain K	R_e	I_m
-19.917	16.50	1.029	-20.323	16.52
-19.917	16.00	.979	-20.308	15.98
-19.933	15.50	.931	-20.308	15.46
-19.948	15.00	.884	-20.292	14.91
-19.948	14.50	.839	-20.292	14.38
-19.964	14.00	.795	-20.277	13.84
-19.964	13.50	.753	-20.277	13.32
-19.980	13.00	.712	-20.261	12.79
-19.980	12.50	.673	-20.261	12.27
-19.995	12.00	.635	-20.245	11.76

Figure 15. Root Locus Comparison.

If, for example, the actual loop gain was 10, then for K = .90, the system would be behaving as one with a loop gain of 9.

One last point worth mentioning is that the root loci of Figure 13 are only a portion of the loci of the Laplace and Z transform systems. Actually, the z transform roots repeat themselves up and down the imaginary axis at multiples of $2\pi/T$. It is only necessary to match the primary roots of the z transform. The other multiple roots will automatically be matched when the primary roots are matched to the Laplace roots.

CONCLUSION

The difference equation technique described in this paper has been successfully applied to aircraft and spacecraft simulations. In these simulations, the difference equations were used primarily to describe the mathematical behavior of both linear and nonlinear control systems. A reduction in the computing time for simulating these control systems was achieved. This reduction ranged from a factor of 15 to 20.

The results of this technique may be made to agree almost exactly with those obtained from standard Runge-Kutta integration techniques merely by increasing the solution rate. However, increasing the rate will increase the computation time which may make the real-time criterion difficult to achieve. The aircraft simulation was solved at a rate of 20 solutions per second. The maximum instantaneous deviation from Runge-Kutta solved at 2,000 solutions per second for the output of the roll, yaw, and pitch channels was about 0.5%. The steady-state values agree almost exactly.

The stability of the simulation is known be-

fore the simulation is actually performed. This preliminary knowledge is gained, of course, from the root locus plot.

The difference equation technique may be applied to a wide range of control problems. Other possible applications for the technique outside the aerospace field are hydrofoil boats, submarines, and industrial control applications.

ACKNOWLEDGEMENT

The difference equation techniques described in this paper were originally developed by Maurice E. Fowler of IBM.

APPENDIX—BASIC z TRANSFORMS

Name of Function	t Domain	S Domain	z Domain
ANY	f(t)	F(S)	F(z)
STEP	u(t)	$\frac{1}{S}$	$\frac{z}{z-1}$
RAMP	t	$\frac{1}{S^3}$	$\frac{Tz}{(z-1)^2}$
PARABOLA	t^2	$\frac{2}{S^3}$	$\frac{T^2(z^2+z)}{(z-1)^3}$
EXPONENTIAL	e^{-at}	$\frac{1}{S+a}$	$\frac{z}{z-e^{-aT}}$
SINE	sin bt	$\frac{b}{S^2+b^2}$	$\frac{(\sin bT)z}{z^2-2(\cos bT)z+1}$
COSINE	cos bt	$\frac{S}{S^2+b^2}$	$\frac{z^2-(\cos bT)z}{z^2-2(\cos bT)z+1}$
DAMPED SINE	$e^{-at}\sin bt$	$\frac{b}{(S+a)^2+b^2}$	$\frac{e^{-aT}(\sin bT)z}{z^2-2e^{-aT}(\cos bT)z+e^{-2aT}}$
DAMPED COSINE	$e^{-at}\cos bt$	$\frac{S+a}{(S+a)^2+b^2}$	$\frac{z^2-e^{aT}(\cos bT)z}{z^2-2e^{-aT}(\cos bT)z+e^{-2aT}}$

BIBLIOGRAPHY

1. FOWLER, M. E., *Numerical Methods for the Synthesis of Linear Control Systems*, IBM TR 24.001.
2. JURY, E. I., *Sampled-Data Control Systems*, John Wiley & Sons, Inc., pp. 1-63.
3. LEONDES, C. T., *Computer Control Systems Technology*, McGraw-Hill, pp. 307-362.
4. RAGAZZINI, J. R. & FRANKLIN, G. F., *Sampled-Data Control Systems*, McGraw-Hill, pp. 1-116.
5. TRUXAL, JOHN G., *Automatic Feedback Control System Synthesis*, McGraw-Hill, pp. 501-557.

DISCONTINUOUS SYSTEM VARIABLES IN THE OPTIMUM CONTROL OF SECOND ORDER OSCILLATORY SYSTEMS WITH ZEROS*

Lt. Cmdr. William B. Nevius, U.S.N.
Norfolk Test and Evaluation Detachment
Norfolk, Virginia
and
Harold Titus
Assoc. Professor of Electrical Engineering
U. S. Naval Postgraduate School
Monterey, California

I. INTRODUCTION

When controlling the performance of a system, it is often desirable to choose the control that will minimize errors in the system and do it in the shortest possible time. A practical matter that must be considered in the optimization in relation to rapid action is the fact that control is of a bounded nature. In a great many important cases, the constraint on the magnitude of the control effort precludes the use of classical variational techniques to design the controller.

In 1956 Pontryagin hypothesized his "maximum principle" which has since been proven a necessary condition for the optimization of linear systems in relation to rapid action.[1] In solving the minimum time problem for linear systems with bounded control, the principle leads to a "bang-bang" form of control law. This implies that the control effort is always being applied at its maximum value. There remains, however, the task of finding the optimum time to switch the control. Pontryagin's method leads to a rule for switching the controller which is a function of the initial conditions in the system adjoint to the one being controlled. Generally these initial conditions are difficult to find.

It is usually helpful to consider the control problem using state space techniques. The coordinates of the space for an n^{th} order system here are a displacement error and its n—1 time derivatives. The space may be divided into two regions each of which is characterized by the control optimal for the trajectories in that region. Optimum switching between the two conditions of the bang-bang control occur on the hypersurface dividing the space. The switching criteria can then be stated as a function of the state space variables.

Of considerable value in finding the switching surface is the system adjoint to the system. The adjoint can be thought of as the system running in reverse time. By plotting trajectories from the origin of the error state space "backwards" in time, with the control satisfying the respective adjoint variables, a surface is generated which may be related to the optimal switching surface in the system state space.

A problem of interest occurs when the system is of such a nature that when control is ap-

* Supported in part by the Office of Naval Research.

plied, a discontinuity appears in one or more of the system states. This may happen when the control is of a bang-bang form and the forward transmission path of the system contains zeros. It could also show up if the control is of such a form that it approximates an impulse to the system. When there are discontinuities in the state space due to switching it is generally no longer possible to write the switching criteria as a function of the state space variables.

One alternative might be to switch the control as a function of time. This may be done effectively when the number of switchings to reach the origin of the error state space is no more than n—1 in an n^{th} order system. Such a restriction limits one mainly to considering only those systems with real, distinct eigenvalues. Large disturbances in lightly damped (oscillatory) systems may require more than n—1 switchings to zero the error states. The most important consideration when controlling as a function of time is the means of implementing the switching logic. To accomplish time dependent control, it is virtually mandatory that a digital computer be inserted in the control loop.

Another approach to the problem is to find a system that reacts identically to the system with zeros except at the points of discontinuity. Control of this parallel system can be stated in terms of the state space variables. This logic can then be used to switch the original plant.

This paper will be an investigation into the latter method. The problem is as follows:

> Given a second order oscillatory system with one zero, find the optimum control for zeroing the errors in the system in minimum time and for zeroing the errors with minimum fuel.

The method of Pontryagin is used to solve the problem. The brief description of the method presented here is based on the work of Rozonoer.[1]

II. PONTRYAGIN'S MAXIMUM PRINCIPLE

Given the system state variables described by n first order differential equations

$$x_i = f_i\,(x,u,t) \qquad i = 1, \ldots\ldots, n \qquad (1)$$

where x is a column vector in phase space and u is a column control vector consisting of r control elements.

The control $u(t)$ must belong to a closed subset U of admissible controls and must be piecewise continuous. The trajectory $x(t)$ in the phase space is uniquely determined by (1) when control $u(t)$ and the initial conditions

$$x(0) = x^0 = \begin{bmatrix} x^0_1 \\ \cdot \\ \cdot \\ \cdot \\ x^0_n \end{bmatrix} \qquad (2)$$

are given.

The control $u(t)$ of a system may be considered optimum under a variety of criteria. A large class of optimization problems may be solved by presenting the criteria in such a way that the solution is attained by minimizing a linear function of the final value of the state space variables. A control must be selected from U that will transfer the system (1) from x^0 to some fixed closed set G of the phase space such that

$$S = \sum_{1}^{n+1} c_i x_i(T) \qquad (3)$$

is a minimum. The constants c_i and the x_{n+1} coordinate are chosen such that minimizing (3) optimizes the system.

In a great many cases optimization of only one of the coordinates of the system is desired. For example, in order to optimize the magnitude of

$$\int_0^T F(x(t),u(t))\,dt \qquad (4)$$

for T and $x(T)$ either fixed or free in a system (1) for $u(t) \epsilon U$, a new variable is introduced:

$$x_{n+1} = \int_0^t F(x(t),u(t))\,dt \qquad (5)$$

$$x^0{}_{n+1} = 0$$

and another differential equation

$$\dot{x}_{n+1} = F(x(t),u(t))$$

is added to (1). The problem of optimizing the integral leads to optimizing $x_{n+1}(T)$ at $t = T$.

Minimizing $x_{n+1}(T)$ in the system (1) with $x_{n+1}(t)$ adjoined is accomplished by putting the problem in functional form (3) and applying the maximum principle to gain the solution. That is

$$S = \sum_{1}^{n+1} c_i x_i(T) = x_{n+1}(T) \qquad (6)$$

is the functional to be minimized. Here it may be seen that $c_1 = c_2 = \ldots, c_n = 0$ and $c_{n+1} = 1$.

A new dependent variable $p(t)$ is now formed such that

$$p_i(t) = -\sum_{1}^{n+1} p_s \frac{\partial f_s(x,u,t)}{\partial x_i} \qquad i = 1, \ldots\ldots, n+1 \qquad (7)$$

The function

$$H = \sum_{1}^{n+1} p_s f_s(x,u,t) \qquad (8)$$

is introduced from which equations (1) and (7) may now be written

$$x_i = \frac{\partial H}{\partial p_i} \quad p_i = -\frac{\partial H}{\partial x_i} \; i = 1, \ldots\ldots, n+1 \qquad (9)$$

The control $u^*(t)$ is said to satisfy the maximum condition if $H(x^*(t),p^*(t),u^*(t))$ reaches an absolute maximum at each time t $(0 \leq t \leq T)$ where $x^*(t)$ and $p^*(t)$ are the values of the variables at time t with $u^*(t) \in U$ controlling. For linear systems of the type discussed in this paper, the necessary and sufficient condition for minimizing

$$S = \sum_{1}^{n+1} c_i x_i(T)$$

optimally with admissible control is that the control satisfy the maximum condition.

To use the maximum principle, H is formed and maximized with respect to $u(t)$. This produces a

$$u^*(t) = \phi(x,p) \qquad (10)$$

which may be used with Equations (9) and the boundary conditions to find $u^*(x)$. If the end point of $x(t)$ is not fixed, it becomes necessary to obtain boundary conditions on $p(t)$ in order to arrive at a solution. The conditions $p(T)$ may be found using a function $F(x) \leq 0$ which describes G and $x^1(T) \in G$, the end point of an optimum trajectory. The form of $p(T)$ will be stated without detailed explanation; however, it may be noticed that at time $t = T$, $p(T)$ is orthogonal to a hyperplane

$$\sum_{1}^{n+1} a_i(x_i - x_i) = 0$$

through the endpoint of the trajectory and directed toward that portion of G where

$$\sum_{1}^{n+1} c_i x_i \leq \sum_{1}^{n+1} c_i x_i(T).$$

The coefficients a_i may be expressed as a linear combination of the c_i and $b_i(x^1(T))$, the latter being coefficients of a hyperplane through $x^1(T)$ bracketting G. Thus

$$p_i(T) = -\lambda c_i - \mu b_i(x^1(T)) \qquad (11)$$

where λ and μ are non-negative numbers one of which may be set equal to unity as it is only the ratio that is important.

Generally, three situations arise as to final boundary conditions.

(i) If $x_i(T)$ are specified for $i = 1,2, \ldots\ldots$, m then these become the boundary conditions for (9).

(ii) If $x_i^1(T)$ are internal points of G for $i\,(1 \leq i \leq n+1)$ then $b_i(x^1(T)) = 0$ and $p_i(T) = -c_i$.

(iii) If $x_i^1(T)$ are boundary points of G for some $i\,(1 \leq i \leq n+1)$ then $F(x(T)) = 0$ and the $p_i(T)$ are as in (11).

When F is differentiable, the bracketting hyperplane through x^1 has coefficients

$$b_i x^1(T)) = \frac{\partial F}{\partial x_i}\bigg|_{x_i} = x_i(T) \qquad (12)$$

If finding the optimum control for minimum transit time another condition must be fulfilled since T is not fixed beforehand. This condition is that $H(T) = 0$.

III. DEVELOPMENT OF SYSTEM EQUATIONS

The equation of a second order system with zeros may be written

$$\ddot{c} + 2\zeta\omega c + \omega^2\dot{c} = a_1u + a.u \qquad (13)$$

where c is the output variable of the system and u is the output of a controller.

This paper is concerned with control of similar systems that are purely oscillatory in nature, i.e., $E\xi = 0$. To facilitate ease of computation in the analysis, Equation (13) is scaled to

$$c + c = a_1u + u \qquad (14)$$

which when written in terms of the Laplace transform of the output variable becomes

$$C(s) = \frac{(a_1s + 1)U(s)}{s^2 + 1} \qquad (15)$$

This system is represented in block diagram form in Fig. 1.

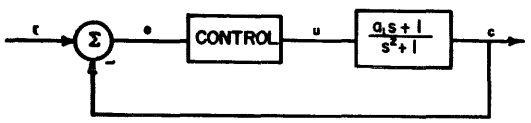


Figure 1. Block Diagram of Control System.

The response of the system to a step input is investigated more readily by means of the error variable

$$e = r - c \qquad (16)$$

If the input r is fed forward as in Fig. 2, the Laplace transform of the error, given

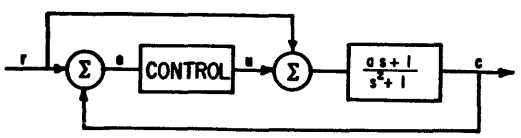


Figure 2. Controlled System with Input Fed Forward.

$$\begin{aligned} c(0) &= c^0 \\ \dot{c}(0) &= \dot{c}^0 \\ R(s) &= r_0/s \end{aligned} \qquad (17)$$

becomes

$$E(s) = \frac{(r_0 - \dot{c}^0)s - (c^0 + a_1r_0) - (a_1s + 1)U(s)}{s^2 + 1} \qquad (18)$$

Now the problem of zeroing the error states reduces to that of zeroing the error initial conditions in the system.

Finally with the introduction of state space variables

$$\begin{aligned} e_1 &= -e \\ e_2 &= \dot{e}_1 \end{aligned} \qquad (19)$$

the system equations can be written in vector matrix notation

$$\dot{e} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} e + \begin{bmatrix} 0 \\ a_1u + u \end{bmatrix} \qquad (20)$$

IV. THE MINIMUM TIME PROBLEM

The problem is stated as follows:

Given the system (20) and a control force of bounded magnitude $|u| \leq 1$, find the optimum control $u^*(t)$ to transfer the state variables from some initial point in the phase space to the origin of the phase space in minimum time T.

That is, given

$$\begin{aligned} e(0) &= e^0 \\ e(T) &= 0 \\ |n| &\leq 1 \end{aligned} \qquad (21)$$

and the system (20), find $u^*(t)$ such that

$$S = \int_0^T \alpha \, dt \qquad (22)$$

is a minimum where α is a positive constant.

Introduce

$$e_{n+1} = e\zeta = S = \int_0^T \alpha \, dt \qquad (23)$$

The system equations then become

$$\dot{e} = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} e + \begin{bmatrix} 0 \\ a_1 u + u \\ \alpha \end{bmatrix} \qquad (24)$$

Because of (21), the functional

$$S = \sum_1^3 c_ie_i(T) = c_3e_3(T) \qquad (25)$$

and since we wish to minimize this, $c_3 = 1$ is chosen. $e_3(T)$ is not limited, hence the boundary condition becomes

$$p_3(T) = -c_3 = -1 \tag{26}$$

By (8), the hamiltonian becomes

$$H = p_1e_2 - p_2e_1 + p_2(a_1u + u) + p_3\alpha \tag{27}$$

Since

$$p_3 = \frac{-\partial H}{\partial e_3} = 0$$

it is evident that p_3 is a constant and therefore $p_3 = p_3(T) = 1$ and now H is

$$H = p_1e_2 - p_2e_1 + p_2(a_1u + u) - \alpha \tag{28}$$

which is maximized in u if

$$a_1u + u = N[\operatorname{sgn} p_2] \tag{29}$$

where $N = \max |a_1u + u|$ for each fixed $t(0 \leq t \leq T)$. The control $u^*(t)$ which satisfies these conditions is a "bang-bang" type control where $u = 1$ at all times and u at the moment of switching is unbounded.

Since e_3 has served its purpose in the optimization process, we may now return to the second order system and solve for the "impulse" variables. By (9),

$$p = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} p \tag{30}$$

and the solution for $u^*(t)$ becomes

$$u^*(t) = 1 \cdot \operatorname{sgn}[\cos(t + \ominus)] \tag{31}$$

where $\ominus$ is a phase angle dependent on x^0.

Several properties of the optimum controller are now known. First, the control is a bang-bang type which applies maximum effort at all times in one of the two "directions." It is switched periodically from one state to the other every half cycle until the origin is reached. Notice that each time the control is switched, a discontinuity appears in the e_2 variable. This occurs because u contains an impulse.

$$\Delta e_2 = \int_{t_s^-}^{t_s^+} (-e_1 + a_1u + u)\,dt = \int_{t_s^-}^{t_s} a_1\,u\,dt = a_1[u(t_s) - u(t_s)] \tag{32}$$

where Δe_2 is the discontinuity in e_2 at the time of switching t_s.

One would now like to find a switching curve $L(e)$ which divides the phase plane e_1 vs. e_2 in such a manner that control $u^* = +1$ is optimum in the space to one side of the curve and $u^* = -1$ elsewhere. Control would be switched when the trajectory $e^*(t)$ crosses the curve. The discontinuity Δe_2 precludes this possibility. For example, examine the trajectory $e^*(t)$ for some initial conditions that dictate $u^* = -1$ for optimum control. At the point where this trajectory crosses $L(e)$ the optimum becomes $u^* = +1$. The control switches and $\Delta e_2 = +2a_1$ occurs which places the states back in the space where $u^* = -1$ was optimum. Here the control switches again, $\Delta e_2, = -2a$, occurs and chatter motion begins. The fact that e_2 is multiple valued at the instant of switching makes a simple realization of $L(e)$ impossible.

For periods between switchings where $u = 0$, the system is well behaved with the solution for the k^{th} interval

$$e_1(t) = K\phi\cos(t + \phi_k) - \delta$$
$$e_1(t) = K\phi\cos(t + \phi_k + \pi/2) \tag{33}$$

where $\delta = 1 \cdot \operatorname{sgn} p_2$ and K, ϕ_k depend on conditions of states at the start of the k^{th} interval.

4.1 *The transformed variable*

The search for a variable of the system on which to control leads to the possibility of "subtracting out" the discontinuity present in e_2 at times of switching.

The Laplace transforms of the system variables are

$$E_1(s) = \frac{e_1s + e_2 + (a_1s + 1)U(s)}{s^2 + 1}$$
$$E_2(s) = \frac{e_2s - e_1 + s(a_1s + 1)U(s)}{s^2 + 1} \tag{34}$$

where

$$U(s) = \delta\left(\frac{1}{s} - \frac{2}{s}e^{-t_1s} + \frac{2}{s}e^{-t_2s} - \cdots\right) \tag{35}$$

which for any instant of time $t(0 \leq t < t_1)$

$$U(s) = \frac{\delta}{s} \tag{36}$$

Equations (34) then become

$$E_1(s) = \frac{e_1^0 s^2 + (e_2^0 + a_1\delta)s + \delta}{s(s^2+1)}$$
$$E_2(s) = \frac{(e_2^0 + a_1\delta)s + (-e_1^0 + \delta)}{s^2+1)} \quad (37)$$

By means of the initial value theorem, it is seen that

$$\lim_{t\to 0} e_1(t) = \lim_{s\to\infty} sE_1(s) = e_1^0$$
$$\lim_{t\to 0} e_2(t) = \lim_{s\to\infty} sE_2(s) = e_2 + a_1\delta \quad (38)$$

At time $t = 0$, e_2 jumps to $e_2 + a_1.\delta$ To remove this discontinuity consider the transformed variables

$$Y_1(s) = E_1(s)$$
$$Y_2(s) = E_2(s) - \frac{a_1\delta}{s} \quad (39)$$

By virtue of (39) and (20)

$$sY_1(s) = sE_1(s) = E_2(s) = Y_2(s) + \frac{a_1\delta}{s}$$
$$sY_2(s) = sE_2(s) - a_1\delta = -Y_1(s) + \frac{\delta}{s} \quad (40)$$

or

$$y = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} y + \begin{bmatrix} a_1 \\ 1 \end{bmatrix} \delta \quad (41)$$

where δ is a unit step function with sign to be determined. The system (41) is identical to that of (20) except for the action at time of switching. It should be noted, however, that care must be taken in assigning final values to the system described by (41) if the two plants are to be controlled in parallel. The final value theorem and (39) gives

$$\lim_{t\to\infty} Y_2(t) = \lim_{s\to 0} sY_2(s) = \lim_{s\to 0} s\left(E_2(s) - \frac{a_1\delta}{s}\right) = -a_1\delta \quad (42)$$

From this it is observed that zeroing the final states in (20) is analagous to zeroing $y_1(T)$ and attaining a final value

$$y_2(T) = -a_1\delta \quad (43)$$

in the system (41).

4.2 *Boundary conditions and final control*

From (38) and (39) it is clear that the initial conditions on the e and y variables are identical. From (39) it is also seen that

$$y_1(T) = e_1(T)$$
$$y_2(T) = e_2(T) - a_1\delta(T) \quad (44)$$

At this point in the pursuit of the optimum control, it becomes necessary to investigate the system action possible at time $t = T$ under admissile control. Δe_2 of (32) provides a means of changing the value of e_2 instantaneously by an amount dictated by the constraints on $u(t)$. With this in mind, it is noted that appropriate use of Δe_2 within the bounds of allowable control may zero the e_2 variable in zero time given that $e_2(T)$ is within range. The conditions (21) and (44) with (32) indicate that for

$$|y_2(T)| \leq a_1 \quad (45)$$

the system (20) may be zeroed instantly.* The boundary conditions on (41) then become

$$y_i(0) = y_i = e_i \qquad i = 1,2$$
$$y_1(T) = 0 \quad (46)$$
$$|y_2(T)| \leq a_1$$

The final controller $u_2(T)$ that must zero the errors for $t > T$ has two conditions imposed upon it, i.e.,

$$a_1u_2 + u_2 = 0$$
$$\dot{u}_2(T) - \delta(T) = \frac{-e_2(T)}{a_1} \quad (47)$$

The solution to (47) is

$$u_2(t) = \frac{-y_2(T)}{a_1} \exp\left(\frac{-t+T}{a_1}\right) \quad t \leq T \quad (48)$$

It is assumed then that $u_2(t)$ is available at time $t = T$ so that the boundary conditions on the system are as stated in (46).

4.3 *Switching functions*

The method of finding a function $L(y)$ with which to describe the switching criteria for the optimum trajectory proceeds as follows. As in the discontinuous case, it is desired that

$$S = \int_0^T \alpha dt \quad (49)$$

* The conditions are stated in terms of the y variable for convenience in order that notational problems arising from multiple value of $e_2(0)$ be avoided.

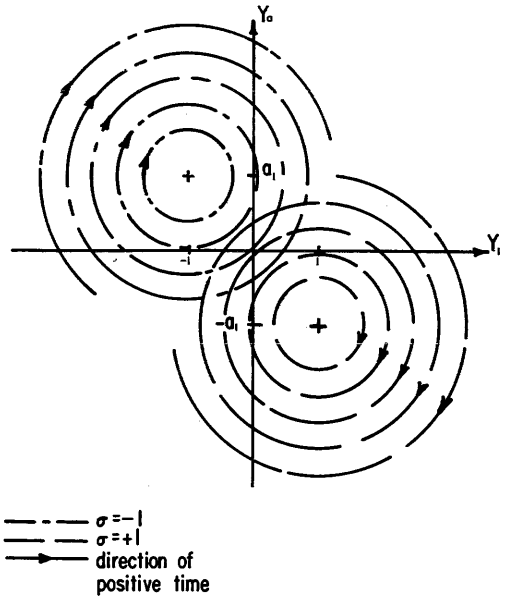


Figure 3. y_1 vs. y_2 Phase Plane with Trajectories for $\delta = \pm 1$.

be minimized, therefore, another variable $y_3 = S = c_3 Y_3(T)$ is adjoined to the system and once again $c_1 = c_2 = 0$. The hamiltonian becomes

$$H = p_1 y_2 + p_1 a_1 \delta - p_2 y_1 + p_2 \delta - \alpha \quad (50)$$

This is maximized in δ when

$$\delta = 1 \cdot \operatorname{sgn}(a_1 p_1 + p_2) \quad (51)$$

With this control, trajectories are circular about $(\delta, -a_1\delta)$ with radius determined by y^0. (See Fig. 3.)

Previous agruments have determined that the conditions on the system are

$$\begin{aligned} y_i(0) &= y_i \qquad i = 1,2,3 \\ y_i(T) &= 0 \\ |y_2(T)| &\leqq a_1 \\ H(T) &= 0 \\ p_3(T) &= -1 \end{aligned} \quad (52)$$

The function

$$F = \frac{1}{2}(y_2 - a_1) \leqq 0$$

may be used to describe G. From this

$$b_2(y_2(T)) = \frac{\delta F}{\delta y_2}\bigg|_{y_2=y_2} = y_2 \quad (53)$$

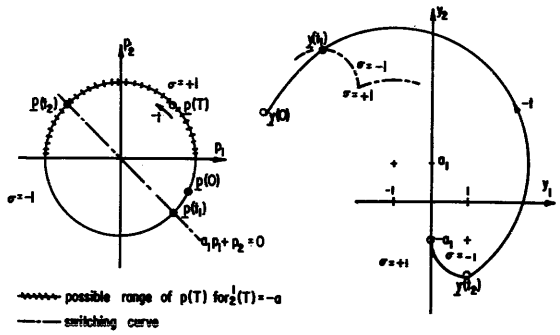


Figure 4. Concurrent Action of $p(t)$ and $y(t)$ for $y_2(T) = -a_1$.

and

$$p_2(T) = -\lambda c_2 - \mu b_2(y_2(T)) = -\mu y_2 \quad (54)$$

where $\mu \geqq 0$ with modulus such that $F(T) = 0$. In the phase plane of p_1 vs. p_2 it is sufficient to note that for trajectories terminating at $y_1(T) = -a_1$, $p_2(T) \geqq 0$ and for trajectories ending at $y_1(T) = a_1$, $p_2(T) \leqq 0$. This information in addition to the control (51) completely define $L(y)$ for trajectories ending on the extremes of the line segment $y_2(T) | \leqq a_1$.

Fig. 4 depicts representative action for optimum trajectories terminating at $y_2(T) = -a_1$, $y_1(T) = 0$. Trajectories ending at $y_2(T) = +a_1$, $y_1(T) = 0$ are mirror images. The optimum switching curves are generated by picking arbitrary values of $p(T)$ from the admissible set for the corresponding boundary values of $y(t)$ and working backwards in time plotting the switching points determined from $p(-t)$ on the y_1 vs. y_2 phase plane.

For trajectories ending in the interior of the line segment where $| y_2(T) | < a_1$, $b_2(y_1(T)) = 0$ and, therefore, $p_2(T) = 0$. This completes the information necessary to describe $L(y)$. Fig. 5 shows a representative trajectory arrived at by translating switching criteria from the p plane to the y plane. Fig. 6 portrays the curve with all dimensions.

V. THE MINIMUM FUEL PROBLEM

The minimum fuel problem is solved by minimizing the integral

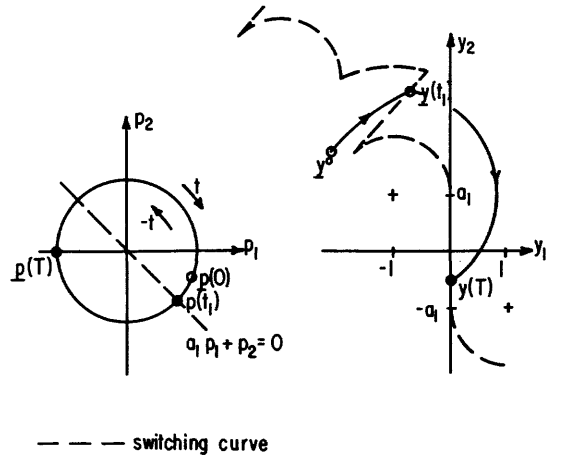


Figure 5. p_1 vs. p_2 and y_1 vs. y_2 Phase Planes with Complete Switching Curves.

$$J = \int_0^T (|u|) + |a_1 u|)dt \qquad (55)$$

in the system (20) where T is not specified. It appears simpler, however, to once again make use of the transformed variable $y(t)$. By minimizing

$$J = \int_0^T |u|dt \qquad (56)$$

in the transformed system (41), the desired result can be obtained provided

i) the switchings in the time interval $0 \leq t \leq T$ are kept to a minimum.

ii) adjustment is made at time $t = T$ when fuel is consumed zeroing the error states $e(t)$ with the exponential control $u_2(T)$.

After adjoining (56) to the system (41), the hamiltonian becomes:

$$H = p_1 y_2 - p_2 y_1 + u(a_1 p_1 + p_2) - |u| \qquad (57)$$

Since T is not specified $H(T) = 0$. With $u(t)$ constrained as before, the control that maximizes H with respect to $p(t)$ is:

$$u^* = 1 \cdot \text{sgn}\,(a_1 p_1 + p_2) \quad |a_1 p_1 + p_2| > 1$$
$$u^* = 0 \quad |a_1 p_1 + p_2| < 1 \qquad (58)$$

5.1 *Initial conditions*

Taking the time derivative of H

$$\frac{dH}{dt} = (a_1 p_1 + p_2)\frac{du}{dt} - \frac{d|u|}{dt} \qquad (59)$$

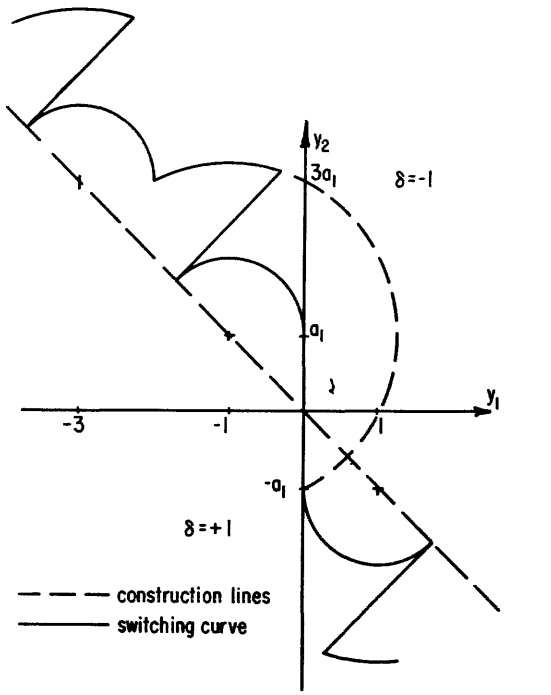


Figure 6. Optimum Switching Curve (minimum time).

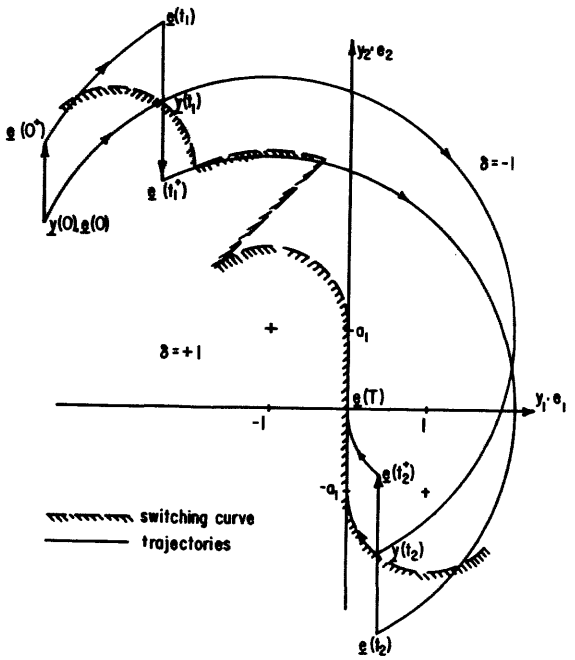


Figure 7. Optimum Trajectories of $e(t)$ and $y(t)$ (minimum time).

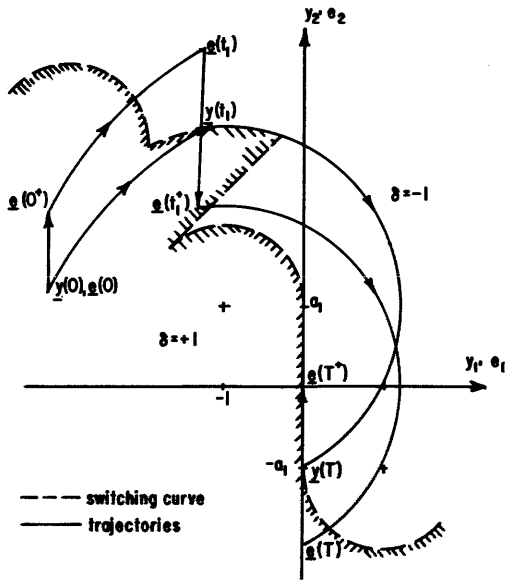


Figure 8. Optimum Trajectories of $e(t)$ and $y(t)$ (minimum time)

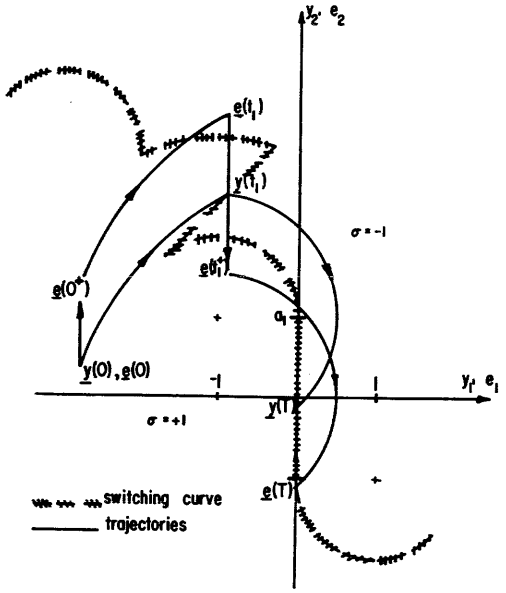


Figure 9. Optimum Trajectories of $e(t)$ and $y(t)$ (minimum time).

it can be seen that $\frac{dH}{dt} = 0$ if $\frac{du}{dt} = 0$. It may also be argued that the change in the hamiltonian with time is zero if

$$a_1 p_1 + p_2 = \frac{d|u|}{du} = \frac{\Delta|u|}{\Delta u} \qquad (60)$$

Since $u(t)$ is switching between $u = 0$ and $u = \pm 1$ and vice versa, this means that the hamiltonian remains constant if the control is switched at $a_1 p_1 + p_2 = 1 \cdot \text{sgn}\ (\Delta u)$. (See Fig. 10.)

By choosing control $u^*(t)$ the hamiltonian remains at its maximum value, i.e., identically zero from time $t = 0^+$ after initial control has been applied until time $t = T$. This control minimizes the integral (56) but does not necessarily minimize total fuel when fuel consumed at switchings is added. In order to minimize switchings, it appears necessary to choose the degenerate case, i.e., $u = 0$ until such time as $a_1 p_1 + p_2 = 1 \cdot \text{sgn}\ (\Delta u)$ where Δu is the change in $u(t)$ when turning the control on. Notice that this choice guarantees that $H(t) = 0$ for all t, $0 \leq t \leq T$. With this in mind, the problem remains to minimize fuel in the non-degenerate case. For this purpose it will be considered that time $t = 0$ is that time when

$$a_1 p_1 + p_2 = 1 \cdot \text{sgn}\ (\Delta u) \qquad (61)$$

and initial control is applied.

At $t = 0$ it may be verified from (61) and because $H(0) = 0$ that

$$p_1 y_2 - p_2 y_1 = 0 \qquad (62)$$

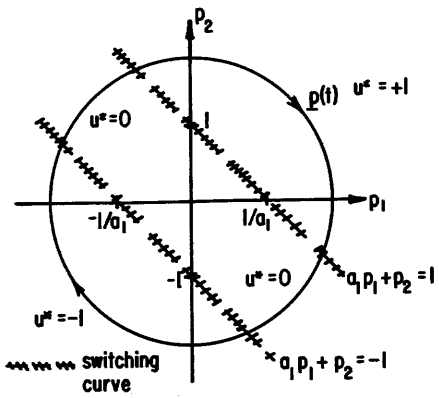


Figure 10. Switching Criteria in p_1 vs. p_2 Phase Plane.

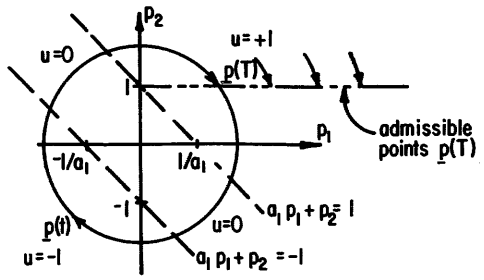


Figure 11. Admissible Points $p(T)$ for $y_2(T) = -a_1$.

5.2 *Final boundary conditions*

In order to investigate final value boundary conditions, the optimum trajectories terminating such that $y_1(T - \Delta t) > 0$ are considered. Trajectories in the rest of the space are mirror images. As in the minimum time problem, an optimum trajectory terminating at $y_2(T) = -a_1$, $y_1(T) = 0$ is investigated first. The determination that $p_2(T) \geqq 0$ as argued in (54) is still valid. This condition on $p_2(T)$ along with the fact that $H(T) = 0$ precludes the possibility of a trajectory terminating as above with $u(T) = -1$. The following cases, however, do apply. Consider

$$H(T) = -a_1 p_1(T) + u(T)\,[a_1 p_1(T) + p_2(T)] - |u(T)| = 0 \qquad (63)$$

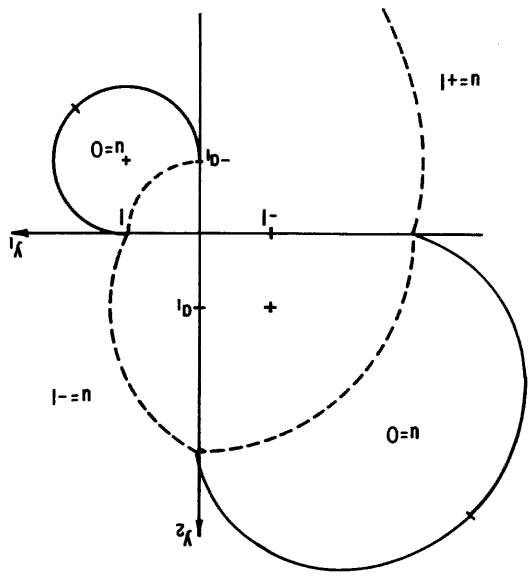


Figure 11a. Switching Criteria for $y_2(T) = -a_1$ (minimum fuel).

This condition implies that if $u(T) = 0$ then $p_1(T) = 0$ and if $u(T) = +1$ then $p(T) \geqq 0$ and $p_2(T) = +1$. Fig. 11 portrays the locus of admissible points $p(T)$ and the switching curves generated by these criteria in the y_1 vs. y_2 phase plane are as in Fig. 11a.

Optimum trajectories terminating on the line segment $y_1(T) = 0$, $|\,y_2(T)\,| < a_1$ must be investigated in a fashion similar to that used with the minimum time problem. Since a final boundary point $y_2(T)$ is not fixed, we may substitute a final condition on $p_2(T)$ to reach a solution. At this point it becomes necessary to decide on the final value functional to be minimized. It is first noted that if the final control to the line segment is $u(T) = 0$, then $-a_1 < y_2(T) < 0$. (It must be remembered that investigation is of trajectories such that $y_1(T-\Delta t) > 0$). If $u(T) = 0$ then also $y_2(T) = e_2(T)$ and in order to minimize the fuel consumed by $u_2(T)$ to zero e_2 after time T then $|\,e_2(T)\,| = |\,y_2(T)\,|$ must be minimized.

If the final control is $u(T) = -1$ ($u(T) = +1$ is not possible for trajectories terminating on this side of the line segment) then $e_2(T) = y_2(T) - a_1$ and, therefore, $|y_2(T) - a_1|$ must be minimized. In both of the above cases, it may be seen that $y_2(T)$ must be maximized on the line segment in order that fuel consumed by $u_2(T)$ to zero the error states be minimized. Therefore, the functional to be minimized is

$$S = \sum_{1}^{3} c_i y_i(T) = -y_2(T) + y_3(T) \qquad (64)$$

where

$$y_3(t) = \int_0^t |u|\,dt \qquad (65)$$

By prior arguments $p_2(T) = -c_2 = +1$ and $p_3(T) = p_3(t) = -c_3 = -1$.

5.3 *Generating the switching curve segments*

It is now helpful to look at the hamiltonian under each of the above conditions, i.e., $u(T) = 0$ and $u(T) = -1$. In the first case

$$\begin{aligned} u(T) &= 0 \\ H(T) &= H(t) = 0 \\ y_1(T) &= 0 \\ p_2(T) &= +1 \\ -a_1 &< y_2(T) < 0 \end{aligned} \qquad (66)$$

and

$$H(T)\ p_1(T)y_2(T) = 0$$

which implies that $p_1(T) = 0$. Fig. 12 shows the switching generated by this condition.

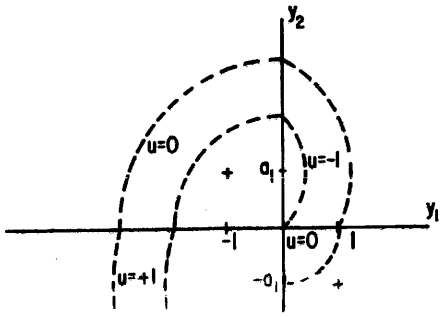


Figure 12. Switching Criteria for $u(T) = 0$ (minimum fuel).

Next is considered the case where

$$\begin{aligned} u(T) &= -1 \\ H(T) &= H(t) = 0 \\ y_1(T) &= 0 \\ p_2(T) &= +1 \end{aligned} \qquad (67)$$

and

$$H(T) = p_1(T)\, y_2(T) - 1\ [a_1 p_1(T) + 1] - 1 = 0$$

from which

$$p_1(T) = \frac{2}{y_2(T) - a_1} \qquad (68)$$

Since $u(T) = -1$ and $p_2(T) = +1$, conditions (58) are met only when

$$p_1(T) < \frac{-2}{a_1}$$

which implies $y_2(T) > 0$. In Fig. 13 these trajectories and switching curves are plotted.

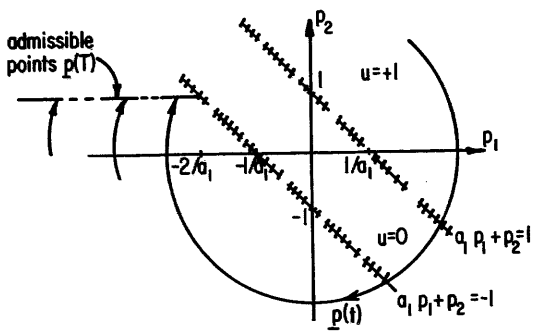


Figure 13. Admissible $p(T)$ where $u(T) = -1$.

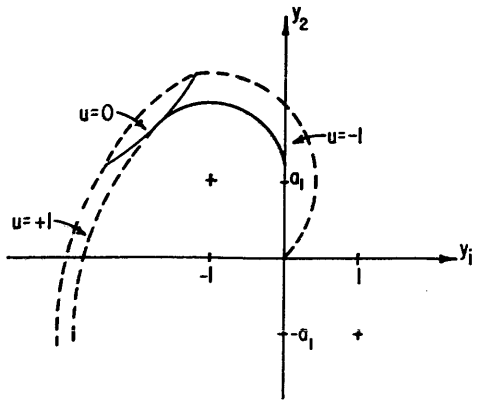


Figure 13a. Switching Criteria where $u(T) = -1$.

5.4 *The complete switching curve*

Because T was never specified and because the fuel consumed at switching was handled as a side condition, a composite of all the calculated switching curves indicates areas in the phase plane where criteria for optimum control appear contradictory. In these areas, analysis by graphical means or actual computation will clear up the situation. Fig. 14 depicts the composite of the first two criteria analyzed.

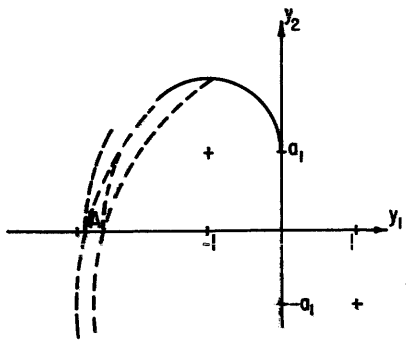


Figure 14. Region of Conflicting Optimum Criteria.

In Fig. 14, region A is an area where there is a question concerning whether it is optimum to switch for $|y_2(T)| = a_1$ or $|y_2(T)| < a_1$. By graphical analysis, it may be seen that it is optimum to switch so that $|y_2(T)| = a_1$.

A similar contradiction between trajectories switching for $0 < y_2(T) < a_1$ and $-a_1 < y_2(T) < 0$ may also be resolved graphically.* The final

* Appendix I presents computational analysis of the resolving process.

result consisting of switching criteria to zero the errors in the system (20) with minimum fuel is given by Fig. 15.

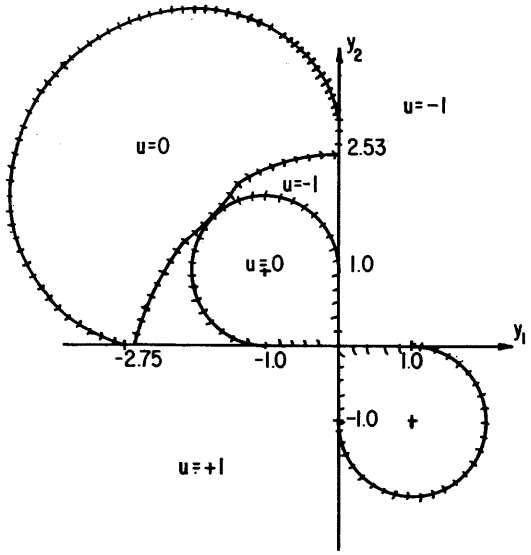


Figure 15. Switching Criteria for Minimum Fuel, $a_1 = 1.0$.

VI CONCLUSIONS

The methods used in this paper to arrive at a solution may be used to good advantage in the investigation of any n^{th} order system with no more than n—1 zeros. The maximum principle provides a powerful tool in optimization, particularly for linear systems. Often the method of Pontryagin will indicate areas of interest to investigate when searching for an optimum control even if the unique solution is not readily forthcoming.

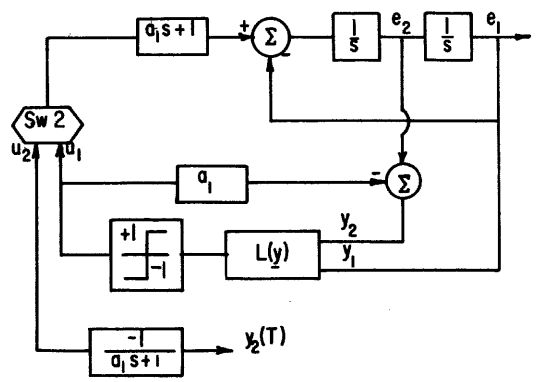


Figure 16. Block Diagram of the Controlled Plant.

The problem of controlling a plant with zeros is analagous to controlling a plant without zeros using an impulse-step type controller. Results obtained in this paper can be adapted to formulate the logic of this type control.

The realization of the true optimum switching logic in a practical system may in many cases not be worth the effort. Quasi-optimum control using simple switching functions that are for the most part linear is a subject for further investigation. Setting time for the system is relatively insensitive to limited variations from the optimum when trajectories are out beyond the first cusp of the switching curve.

APPENDIX I

In the past, the electrical engineer, has usually turned to the analog computer for problem solving and his reference of familiarity is strongest there. To test the drift of the ampli-

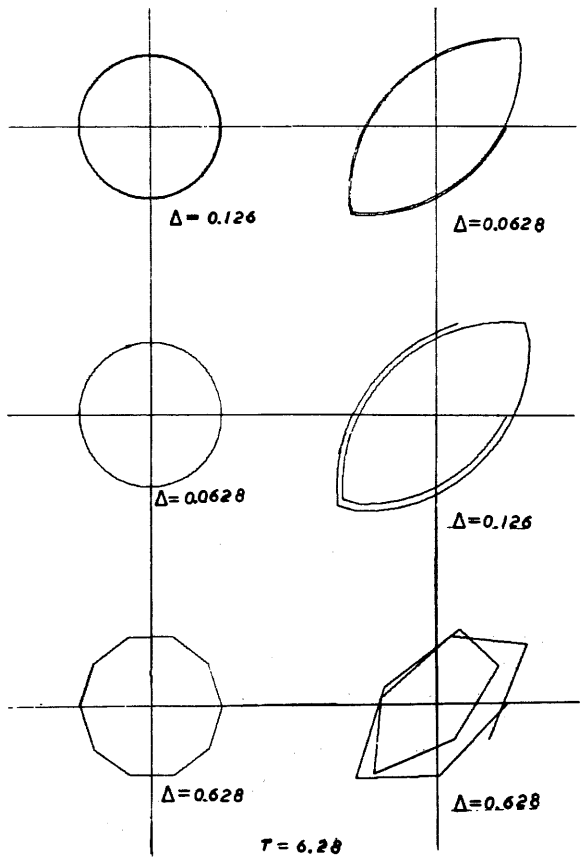


Figure A–1. Trajectories showing the effects of different integration step sizes (△) on the Runge-Kutta method. Period of unforced system being integrated was T = 6.2836.

fiers, a common method was to plug in a simple oscillatory system

$$\dot{y} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} y$$

and to study the decay of the circular trajectory in the phase plane (y vs. ẏ).

This same procedure was used here to study the different integration schemes used and to determine the proper integration step size. The forcing function ($u = \pm 1$) was introduced to give breaks in slopes and also discontinuities in the steady state limit cycle described by

$$\dot{y} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} y - \begin{bmatrix} 1 \\ 1 \end{bmatrix} \text{sgn } (y_1 - y_2)$$

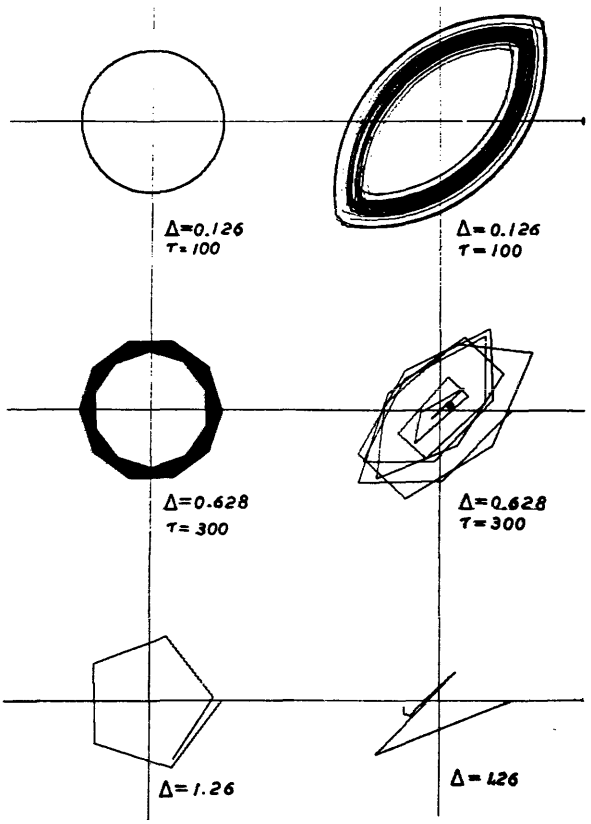


Figure A–2. Trajectories showing the effects of different integration step sizes (△) on the Runge-Kutta method. Period of unforced system being integrated was T = 6.2836.

This graphical means of analyzing gave considerable insight in evaluating the integration schemes. The following figures give a brief résumé of these studies.

The 4th order Runge-Kutta method was found to be the most foolproof when dealing with discontinuities in the state variables. In the continuous variables the Adams Bashforth predictor-corrector[4] with a Taylor expansion or Runge-Kutta starter were found efficient and useful.

The system was also tested with the plant on the analog computer and the control with its nonlinear switching function being generated on an on-line digital computer.

BIBLIOGRAPHY

1. ROZONOER, L. I., "L. S. Pontryagin's Maximum Principle in the Theory of Optimum Systems—Part I," Avtomatika i Telemekhanika, Vol. 20, pp. 1288-1302, October 1959.

   ROZONOER, L. I., "L. S. Pontryagin's Maximum Principle in Optimal System Theory —Part II," Avtomatika i Telemekhanika, Vol. 20, pp. 1405-1421, November 1959.

2. FLUGGE-LOTZ, I., and TITUS, H. A., "Optimum and Quasi-optimum control of third and fourth-order systems," Division of Engineering Mechanics, Stanford University Technical Report No. 134, pp. 8-12, October 1962.

3. FLUGGE-LOTZ, I., and ISHIKAWA, T., "Investigation of Third Order Contactor Control Systems with Zeros in Their Transfer Functions," NASA TN D-719, January 1961.

4. HAMMING, R. W., "Numerical Methods for Scientists and Engineers," McGraw-Hill Book Company, Inc., New York, 1962.

TWO NEW DIRECT MINIMUM SEARCH PROCEDURES FOR FUNCTIONS OF SEVERAL VARIABLES

Bruno F. W. Witte Design Specialist
and
William R. Holst Senior Research Engineer
General Dynamics/Astronautics
San Diego 12, California

(1) LINEAR SEARCH METHOD FOR n VARIABLES

(1.1) *Summary Of The Method*

The method can somewhat vaguely be classified as a modified "steepest descent." It is, of course, an iterative procedure. Each cycle, to be iterated on, consists essentially of two parts: in Part I a "best" line is found, in Part II an attempt is made to minimize the given function along this line. Thus, each cycle resembles the corresponding cycle in the method of steepest descent. The method of steepest descent differs from our method in the manner in which in Part I the "best" line is found. Steepest descent, in fact, implies that this line (let us call it the "baseline" from now on) be defined by the starting point for the cycle and the gradient of the function at this starting point, where the starting point, in turn, is the minimum point of the preceding cycle. Well known modifications of the steepest descent are concerned with, for example, baselines restricted to a subspace normal to the preceding baseline, or with different ways of minimizing along a given baseline, or perhaps with the question as to how feasible it is to seek a minimum at all along a given baseline during each cycle before switching to the next baseline.

(1.2) *Symbol Explanations*

y — The ordinate of arbitrary points in the space Φ, i.e. that coordinate which is plotted in the same direction as is the function value.

F — The function to be minimized.

n — Number of independent variables in $F = F(x_1 \ldots x_n)$.

S — Designates the hyper-surface defined in Φ by F.

Φ — Designates the space spanned by the n independent variables $x_1 \ldots x_n$, the abscissas, and by y, the ordinate.

ϕ — The space spanned by the n independent variables $x_1 \ldots x_n$.

x_i — The abscissas, i.e. the independent variables, $i = 1 \ldots n$.

x — The point with coordinates x_i, i.e. $x = (x_1 \ldots x_n)$.

x^k — The starting point in ϕ for minimizing F along b^k in cycle C^k. Also the minimum point found by minimizing F along b^{k-1} in cycle C^{k-1}. $x_1^k \ldots x_n^k)$, $k = 0, 1, 2, \ldots$.

C^k — Designates the k-th cycle of the iterative procedure leading to the minimum of F. $k = 0, 1, 2, 3, \ldots$.

F^k — An abbreviation for $F(x^k)$.

S^k — The starting point in Φ corresponding to x^k, i.e. $S^k = (x^k, F^k)$, $k = 0, 1, 2, \ldots$.

T^k — The n-dimensional hyper-plane tangent on S in S^k.

b^k — The baseline chosen in cycle C^k.

grad — An n-vector in the space ϕ pointing in the direction of steepest acscent of any subspace of Φ or manifold of Φ with dimension 1, 2, . . . , or n. The magnitude of this vector is equal to the slope of steepest ascent. For linear subspaces this vector is independent of x. For the subspace S this vector is the familiar gradient of F.

Λ — An operator designating the intersection of the two subspaces between which it stands.

P^i — A subspace of Φ with dimension n — i defined by $P^i = T^0 \Lambda T^1 \Lambda \ldots \Lambda T^i$.

Q^i — A straight line in Φ defined by $Q^i = T^{i-n+1} \Lambda T^{i-n+2} \Lambda \ldots \Lambda T^i$, $(i \geqq n-1)$.

p^i — The intersection $P^i \Lambda \phi$.

q^i — The point $Q^i \Lambda \phi$.

L^i — The point on Q^i with $y = 1$.

^{i} — The projection of L^i on ϕ.

E^k — The equation of a hyper-plane parallel to T^k through the origin of Φ.

λ_j — Lagrangian undetermined multipliers.

η^k —
ξ^k — $\xi^k = (\xi^k_1 \ldots \xi^k_n)$ } The running coordinates of an arbitrary point on T^k.

a^k_i — Defined by $a^k_i = -\partial F/\partial x_i$, evaluated at x^k.

s — The distance from x^k to a point on b^k. s is positive in the downward direction.

G — The function F considered as a function of only s, i.e. $G(s) = F(x_1(s), \ldots, x_n(s))$.

G′ — The derivative dG/ds.

(1.3) *The Baseline Choice*

The method used to choose the next baseline can best be visualized as follows: The function $y = F(x_1 \ldots x_n)$ defines an n-dimensional hyper-surface, S, embedded in the (n + 1) — space, Φ, spanned by the n-spaced, ϕ, of the independent variables $x_1 \ldots x_n$ and the dependent variable y. Let $x^0 = (x^0_1 \ldots x^0_n)$ be the starting point of the first cycle, C^0, of our search procedure, and let $F^0 = F(x^0)$ and x^0 define the corresponding starting point $S^0 = (x^0, F^0)$ on the hyper-surface.* Let T^0 designate the hyperplane tangent in S^0 on S (whenever necessary we will assume that such tangent hyperplanes exist). The baseline, b^0, for the first cycle, C^0, is then chosen to be the line through x^0 and with direction defined by the gradient of T^0, grad T^0. Naturally, we have for this cycle grad T^0 = grad $F(x^0)$. We will now assume for the moment that we know how to proceed with part II, and that at the end of part II we obtain an approximation $x^1 = (x^1_1 \ldots x^1_n)$ for the minimum of S along b^0. x^1 is also the starting point of cycle C^1. (In general, x^i designates the starting point and s^{i+1} the minimum point of cycle C^i; $S^i = (x^i, F^i)$ is the point on the hyper-surface corresponding to x^i, and T^i is the hyperplane tangent in S^i on S; finally b is the baseline for cycle C^i). The next baseline b^1 is then chosen to be the line through S^1 and with direction defined by grad $(T^0 \Lambda T^1)$, where $T^0 \Lambda T^1$ is the intersection of T^0 and T^1, i.e. an (n—1)-dimensional linear subspace of Φ. Assuming again that we know how to find x^2, the minimum point along b^1, b^2 is found as the line through x^2 in a direction given by grad $(T^0 \Lambda T^1 \Lambda T^2)$.

In general, for $i < n-1$, b^i is determined so that it becomes the baseline through x^i with a direction given by grad $(T^0 \Lambda T^1 \Lambda \ldots \Lambda T^i)$. The argument of the gradient, $P^i = T^0 \Lambda T^1 \Lambda T^2 \Lambda \ldots \Lambda T^i$, is a linear subspace of Φ, and has dimension n—i. For $i = n-1$, P^i would then be a straight line; for $i = n$, it would be only one point, for $i > n$, P^i is an empty set; for $i \geqq n$, therefore, grad P^i is not defined. For $0 \leqq i < n-1$, grad P^i is defined, of course, as the direction in ϕ along which P^i ascends most rapidly. Since all P^i are linear, grad P^i is independent of x.

For $i \geqq n-1$, b^i is determined so that it becomes the baseline through x^i with a direction given by grad Q^i, where $Q^i = T^{i-n+1} \Lambda T^{i-n+2} \Lambda \ldots \Lambda T^i$. Thus, all Q^i are straight lines in Φ, and their gradients are parallel to their normal projections on ϕ.

*An attempt is made in the above and subsequent notation to designate by lower-case letters all points, or point sets, or subspaces, which are completely embedded in the n-space of the independent variables $x_1 \ldots x_n$, and to use capital letters if they are not completely contained in this n-space.

In general, for all i, the above defined gradients are determined as follows: Let p^i and q^i be defined by $p^i = P^i \Lambda \Phi$ (for $0 \leqq i < n-1$), $q^i = Q^i \Lambda \phi$ (for $n-1 \leqq i$). Then all q^i are single points. Next consider the point L^i ($i \geqq n-1$) with ordinate $y=1$ on one of the lines Q^i, and its projection ℓ^i on ϕ. The directed line from q^i to ℓ^i is parallel to grad Q^i, and $|\text{ grad } Q^i| = 1/|q^i - \ell^i|$. The n coordinates of $\ell^i = (\ell^i_1 \ldots \ell^i_n)$ are found by solving simultaneously the n linear equations, E^k, corresponding to the n T^k in the expression for Q^i and setting $y = 1$. Setting $y = 1$ is arbitrary, and insures only that $y \neq 0$. The n coordinates of $q^i = (q^i_1 \ldots q^i_n)$ are found by solving the same system with $y = 0$. Next consider some arbitrary point L^i ($0 \leqq i < n-1$) with ordinate $y = 1$ on one of the linear subspaces P^i of dimension $n - i$, and its projection ℓ^i on ϕ. Let q^i (for $0 \leqq i < n-1$) be the point on p^i closest to ℓ^i. The directed line from q^i to ℓ^i is then, again, parallel to grad P^i, and also $|\text{ grad } P^i| = 1/|q^i - \ell^i|$. Since L^i can be chosen arbitrarily from all the points on P^i with $y = 1$, we set $\ell^i_k = 0$ (for $k = i+2, \ldots, n$), and find the first $i + 1$ coordinates of $\ell^i = (\ell^i_1 \ldots \ell^i_{i+1}, \ell^i_{i+2} \ldots \ell^i_n)$ by solving simultaneously the $i+1$ linear equations, E^k, corresponding to the $i+1$ T^k in the expression for P^i, again also setting $y = 1$. The n coordinates of $q^i = (q^i_1 \ldots q^i_n)$ are determined by requiring that the square of the distance from ℓ^i to q^i be a minimum, i.e. $d^2 = (\ell^i_1 - q^i_1)^2 + (\ell^i_2 - q^i_2)^2 + \ldots + (\ell^i_n - q^i_n)^2 = \min$. Furthermore, we note that the q^i_k must satisfy the constraint that q^i is to be on p^i, which is the same as saying that the q^i_k must also satisfy the $i + 1$ linear equations, E^k ($k = 0 \ldots i$), corresponding to the $i + 1$ T^k in the expression for P^i, but now setting $y = 0$. The Lagrangian method of undetermined multipliers leads us then to the unconstrained minimization of the function $g(q^i_1 \ldots q^i_n, \lambda_0 \ldots \lambda_i) = d^2 + \lambda_0 E^0 + \ldots + \lambda_i E^i$. Equating to zero the partial derivatives of g with respect to the n q^i_k and the $i + 1$ λ_j, we obtain again a system of linear algebraic equations, here of order $n + i + 1$. Note that this way of finding q^i applies only to the initial cycles of the search for which $i < n - 1$; hence the largest linear system to be solved here will have order $n + (n-2) + 1 = 2n-1$.

The linear equations E^k ($0 \leqq k$) correspond to our hyperplanes T^k which are tangent to the hyper-surface S at the locations x^k. The analytic expression for one of the E^k can readily be obtained from a Taylor expansion of F about x^k: $F(x) = F(x^k) + (x - x^k) \cdot \text{grad } F(x^k) +$ higher-order terms. If we replace $F(x)$ by η^k and $x = (x_1 \ldots x_n)$ by $\xi^k = (\xi^k_1 \ldots \xi^k_n)$, delete the higher-order terms, collect all constant terms, and rewrite this expansion in terms of the individual running coordinates on the tangent plane, we obtain with grad $F(x^k) = -(a^k_1 \ldots a^k_n)$: $a^k_1 \xi^k_1 + a^k_2 \xi^k_2 + \ldots + a^k_n \xi^k_n + \eta^k = b^k$. This can be simplified considerably for computational purposes by always setting $b^k = 0$. This is permissible because we are only interested in the slopes of the T^k, or in the slopes of subspaces common to several T^k. The E^k are then explicitly: $E^k = a^k_1 \xi^k_1 + \ldots + a^k_n \xi^k_n + \eta^k = 0$, and they describe hyperplanes which are parallel to the tangent planes, and contain the origin of the space Φ. The a^k_j could be evaluated directly at x^k as the partial derivatives of F with respect to x_j. Or they may be approximated by finite difference approximations to these derivatives. The computer program referred to in the abstract uses first-order difference approximations.

(1.4) *Minimizing along the Baseline*

The search for the baselines was explained in the preceding section. In this section we describe what to do with a baseline once it is selected, i.e. how to minimize the given function $F(x_1 \ldots x_n)$ along this line. Let us introduce an (s, G)-coordinate system with origin at x^k, with the positive s-axis in the downward direction of the baseline, and with the positive G-axis in the positive y-direction. The function G(s) is defined as $G(s) = F(x_1(s) \ldots x_n(s))$. We restate now our problem more precisely: given $s_0 = 0$, $G_0 = G(s_0)$, $G_0 = G'(s_0)$, and $\bar{s}_1$ (the initial step size), find a "bracketed" approximation, s_m, to a relative minimum of G(s), and verify the existence of two bracketing numbers s_a and s_b, such that $s_a < s_m < s_b$ and $G(s_a) > G(s_m) < G(s_b)$. The procedure is described below in several steps.

STEP (1)

s_1 is equated to $\bar{s}_1$, and $G = G(s_1)$ is evaluated. This defines the point $P_i = (s_1, G_i)$ for $i = 1$, and with $G_i = G(s_i)$.

Question: Is $G_1 \geqq G_0$?

If yes, a parabola is passed through P_0 with the given slope G_0, and through P_1. This is done in the following Step (2).

If no, we go directly to Step (3) to check whether P_1 is above or below the line through P_0 with slope G_0.

STEP (2)

The minimum location s_2 is found for the parabola through P_0 and P_1 with slope G_0 at P_0. s_2 will satisfy: $|s_0| < |s_2| \leqq |s_1|/2$. $G_2 = G(s_2)$ is evaluated.

Question: Is $G_2 < G_0$?

If yes, we have solved our problem, i.e. we have obtained a bracketed approximation to the minimum, with $s_a = s_0$, $s_m = s_2$, $s_b = s_1$. However, our experience showed that the expenditure of one more function evaluation often allowed a considerable improvement of the value for s_m. This is particularly desirable toward the end of our search for the minimum of $F(x_1 \ldots x_n)$. The improvement of s_m is obtained by cubic interpolation, and is described in Step (4). Before going there, however, the indices 1 and 2 are interchanged on the quantities P_1, G_1, s_1 and P_2, G_2, s_2 so as to number the points P_0, P_1, P_2 in the order of increasing $|s|$-values.

If no, we substitute P_2 for P_1, and repeat Step (2) until either $G_2 < G_0$, or $|s_2| < |\bar{s}_1|/100$. In the latter case we replace $\bar{s}_1$ by $-\bar{s}_1$, G_0 by the slope of the line (P_0, P_2), and return to Step (1). This starts a search for the minimum of G in the range of negative s-values (assuming that G_0 was inaccurate). In case our search should then again lead us back to the vicinity of the same s-value, $s = s_0$, we set $s_m = s_0$ with brackets $s_a = -|\bar{s}_1|/100$ and $s_b = |\bar{s}_1|/100$. In this case our starting point and our approximation to the minimum are one and the same point. That situation will always happen at the final minimum of $F(x_1 \ldots x_n)$, but may also happen occasionally sooner. The termination procedure, described further below, then is triggered.

STEP (3)

Question: Is P_1 above the line through P_0 with slope G_0 ?

If yes, the minimum location s_2 is found for the parabola through P_0 and P_1 with slope G_0 at P_0. It satisfies $|s_2| > |s_1|/2$. If $|s_2| < |s_1|$, the indices 1 and 2 are interchanged.

If no, the minimum location s_2 is found for the cubic with clope G_0 at P_0 and with its point of inflection at P_1. It satisfies $|s_2| > 2|s_1|$.

Question: Is $G_1 < G_2 = G(s_2)$?

If yes, we found a bracketed approximation to s_{min}, and proceed with Step (4).

If no, we re-define G_0 to be the slope of the parabola through P_0, P_1, P_2 at point P_1, then subtract 1 from all index values occurring in $P_i = (s_i, G_i)$ $(i = 0, 1, 2)$, and return to Step (3).

STEP (4)

A cubic is passed through P_0 with slope G_0, and through P_1 and P_2. $\bar{s}_m$ is the location of the minimum of this cubic. Note that the existence of brackets for $\bar{s}_m$ guarantees the existence of the minimum.

STEP (5)

We test next the *three* lowest values of G, found so far, whether they agree with each other within a given tolerance.

If yes we set s_m equal to that s_i-value for which $G(s_i)$ is smallest. ($\bar{s}_m$ usually is, but need not be, that s_i-value.)

If no, a parabola is passed through the lowest three points, and the minimum value of this parabola is compared with the two lowest available values of G. If these three values agree with each other within the given tolerance we have found s_m; if they do not agree, we evaluate G for that s-value for which the parabola had its minmium, and go to Step (6).

STEP (6)

We test the *two* lowest values of G with the minimum value of the last parabola. If these three values agree with each other within the given tolerance we have found s_m; if they do not agree, a parabola is passed through the lowest three points of G and we compare the minimum values of the last *two* parabolas with the lowest value of G.

If these three values agree with each other within the given tolerance, we have found s_m; if they do not agree, we evaluate G for that

s-value for which the last parabola had its minimum, and repeat Step (6).

(1.5) *Detection of the Minimum*

The iterated execution of the procedures for Parts I and II of each cycle may be discontinued whenever one of the following conditions is detected:

Condition A: The partial derivatives of $F = (x_1^k \ldots x_n^k)$ satisfy $|\partial F/\partial x_i^k| \leqq \epsilon$ for all $i = \ldots n$, and for a given small value of ϵ, say $\epsilon = 10^{-6}$. As explained at the end of section (1.3, these partial derivatives are evaluated to obtain the coefficients a^k in the equations E^k.

Condition B: The starting point and the approximation to the minimum point on the baseline are one and the same point. How this can happen was explained in Step (2) of section (1.4).

Condition C: The matrix of the linear equations E^k, discussed in section (1.3), is very ill-conditioned or singular. This may occasionally happen before either Condition (A) or Condition (B) is satisfied.

Condition D: The numerical values of the coordinates of the minimum location and/or the minimum of the function have stabilized to a given number of significant figures.

Whenever one, or perhaps two of the above conditions are satisfied, there is an excellent chance that the minimum has been found. One may then do one of two things: (i) terminate the execution of the program, or (ii) begin a somewhate more extensive testing procedure in the course of which the vicinity of the suspected minimum location is explored, and it is thereby made even more likely that the solution has been found. This is what was programmed at G.D./Astronautics. Preference was given to method (ii) because it also provides an answer to the question how sensitive the function is to small deviations from the minimum location. The knowledge of this sensitivity is usually desired together with the minimum location itself.

(2) CIRCULAR ARC SEARCH METHOD FOR 2 VARIABLES

(2.1) *Summary of the Method*

The iterative circular search is preceded by a starting procedure, which finds five points near the spine of the valley of the function to be minimized. Thereafter, each iteration begins with an attempt to approximate by a portion of a circle the course of the projection of the valley onto the space of the independent variables. The first circle is fit to three of the five valley points found. All succeeding circles are fit to four or more valley points. As the iteration progresses toward the minimum these circles approach the osculating circle at the minimum location. During each iteration the determination of a circular arc approximation to the spine of the valley is followed by an attempt to predict a point on this arc which is closer to the desired minimum than any other point obtained so far. Again during each iteration, the prediction of such a point is followed by a steepest descent from this point into the valley, which amounts to a minor, yet important correction. The correction is important since it insures that deviations between the courses of the circular arcs and the portions of the spine of the valley approximated by the arcs will not accumulate as the iteration progresses. The new valley point then replaces the highest of the four lowest valley points found before, and this predictor-corrector type method is repeated in the next iteration. Empirical results show that the sequence of valley points obtained converges quite rapidly to the desired minimum. In the following sections we emphasize again the geometric aspects of the more important steps summarized above, rather than to give the algebraic details. This is done since some of the details are still subject to experimentation and analysis.

(2.2) *The Starting Procedure*

We assume that a first guess is available as a starting location x^1. We find its valley point v^1. A valley point v^i will always mean the location of the minimum of the given function F along a line through x^i in the direction of $-\text{grad } F(x^i)$. We find the distance $d^1 = |v^1 - x^1|$ and a unit vector e^1 in the direction of the projection of $-\text{grad } F(v^1)$ on a line normal to $-\text{grad } F(x^1)$. We then find four more valley points v^i $(i = 2 \ldots 5)$ by taking a step $d^i = (3/4)^{i-2}d^1$ in the direction of e^{i-1} from v^{i-1} to x^i, and then by minimizing from x^i to v^i. The directional minimizations are done by using the

same method which was described in section (1.4).

(2.3) *The Approximating Circular Arcs*

The circular arcs are obtained as arcs of "best fitting" circles to the locations v^i of the lowest three, four, or five valley points. Three valley points are chosen only once, i.e. for the first circle; four are usually chosen thereafter; if it happened, however, that the last valley point found had a function value higher than the highest of the four lowest valley points previously known, then five points were chosen.

(2.4) *The Predictor-Corrector Step*

A "best fitting" two-dimensional plane E_2 is found for the valley points $V^i = (v^i, F^i)$ in space Φ, where v^i and i were explained above, and $F^i = F(v^i)$. It is helpful to visualize now a cylinder in Φ parallel to the y-axis and intersecting ϕ along the circle of the preceding section. This cylinder and the above plane E_2 intersect in an ellipse. The location u of the point on this ellipse with the smallest y-value is our predicted point. The distance of the point u from the spine of the valley decreases rapidly as the iteration progresses, and appears to become insignificantly small long before the minimum is reached. Nevertheless, it is unknown at the present time how damaging the cumulative effect woud be. We are employing, therefore, during all iterations a straight steepest descent correction which leads from u to the next valley point. The correction is done in the same way as was explained for the five starting points in (2.2), i.e. by using the method described in (1.4). The initial step taken in minimizing from the low point on the circle to the valley is set equal to one-half the length of the gradient vector at the low point divided by the coefficient of the second-degree term of the parabola approximating the shape of the function along the previous line of minimization. We expect that further studies of this particular correction will lead to considerable savings in the number of function evaluations. For example, it appears that the correction usually need be made only in a radial direction rather than the negative gradient direction, while the latter is needed only occasionally. Moreover, it is doubtful whether any such correction is needed at all in each iteration.

(2.5) *The Switch to Parabolas*

The above predictor-corrector step is taken at least four times. If thereafter the function value of the lowest valley point is within ½% of the function value of the second-lowest valley point, the method of predicting a new point u on the circle is modified somewhat. Instead of finding the above described plane E_2, we find the "best-fitting" line L to the same four points v^i defining our circular arc and in the same plane with this circular arc. The lowest three v^i are then projected on L to give the three points w^i on L. These and the F^i —values associated with the v^i define three points $W^i = (w^i, F^i)$ in the plane through L and parallel to the y-axis. The location w^m on L of the minimum of the parabola through the three points W^i is then projected back onto our circular arc to give our predicted point u. The steepest descent correction is gain applied to u, and produces the next valley point.

(2.6) *The Tail Correction*

After successive approximations to the minimum value of the function have stabilized to five figures, the search terminates with a simple tail correction, which consists in minimizing the function along the line through the two lowest valley points, and which usually adds several significant figures to the answers.

It is interesting to observe the behavior of the radii of the circular arcs after the iteration has begun to converge. As was said before in Section (2.1) the circles approach the osculating circle of the spine of the valley at the minimum location. However, once the sequence of valley points has converged to the minimum, all subsequent points fall very close to each other; they then no longer define the course of the valley nor do they define its osculating circle. Instead, the radii of the "best-fitting" circles *suddenly* shrink by several orders or magnitude to dimensions comparable to the inaccuracies in the coordinates of the minimum location. When this shrinkage occurs the search should be terminated. Usually, however, this does not happen before the search had already been terminated with the above described tail correction.

(2.7) *An Extension to n Variables: The Spherical Shells Method*

The good results obtained with the two-

dimensional Circular Arcs Method suggest its extension to n independent variables as follows:

(—1) START. A starting point x^i is given in n space as a first approximation. Find its valley point v^1, the distance $d^1 = |v^1 - x^1|$, the n — 1 space E^1 normal to grad $F(x^1)$, and the direction e^1 in E^1 as the projection of —grad $F(v^1)$ on E^1. Find n + 3 more initial valley points v^i ($i = 2 \ldots n + 4$) by taking steps $d^i = (n + 1), (n + 2))^{i-2}d^1$ from v^{i-1} in the direction e^{i-1} to x^i, and then by minimizing F in the direction of —grad $F(x^i)$ from x^i to v^i, for all values of $i = 2 \ldots n + 4$.

(— 2) FLATS. Fit an m flat (i.e. a linear m space) to the lowest m + 2 valley points in n space, such that its dimension m is as small as feasible. (The feasibility could be judged by observing successive standard deviations σ_m of the points from the m flats for the sequence of decreasing m — values: m = n, n — 1, . . . , 1. A sudden large increase of σ_{k-1} over σ_k would indicate that m = k is feasible while m = k—1 is not feasible.) Skip to (— 4) if m = 1.

(— 3) SPHERES. Fit the surface of an m-dimensional sphere to the projections of the lowest m + 2 valley points on the above m flat. Some shell of this sphere will approximate the course of the (hyper —) valley. Pin-point next an optimum surface point on this shell to which to go next, find its valley point, and return to (—2).

(— 4) PARABOLAS. Find the parabola through the projections of the lowest three valley points on the plane spanned by the function axis and the regression line found in (—2). Go to the minimum of this parabola, find its valley point, and return to (— 2) unless the minimum value of the function has begun to stabilize.

(— 5) TAIL. Minimize the function along a line through the lowest two valley points in n space. Find the valley point corresponding to this directional minimum. Repeat (— 5) till the function minimum has stabilized to the desired number of figures.

The above described Spherical Shells Method will be programmed at General Dynamics-Astronautics in the near future, and results will be published when available.

(3) EXAMPLES AND COMPARISONS

The following functions of x and y were minimized. ($r = (x^2 + y^2)^{1/2}$ and $\theta = \tan^{-1}(y/x)$.)

$$
\begin{aligned}
&(3.1)\ \text{ROSIE} &&= 100(y - x^2)^2 &&+ (1 - x)^2\\
&(3.2)\ \text{SHALOW} &&= (y - x^2)^2 &&+ (1 - x)^2\\
&(3.3)\ \text{STRAIT} &&= (y - x^2)^2 &&+ 100(1 - x)^2\\
&(3.4)\ \text{CUBE} &&= 100(y - x^3)^2 &&+ (1 - x)^2\\
&(3.5)\ \text{DOUBLE} &&= 100(r - 1)^2(r - 2)^2 &&+ 0.1(2 - x)^2\\
&(3.6)\ \text{HEART} &&= 1000(1 + \cos\theta - r/10)^2 + y^2(x - 10)^4
\end{aligned}
$$

All were programmed in double-precision for an I.B.M. 7094 computer. Results are listed in the following tables, and the results for ROSIE are discussed in detail and compared with other methods.

(3.1) ROSIE

Minimum (F = 0) at (1, 1), with a steep valley along $y = x^2$, and a side valley along the negative y-axis.

Table (3.1A) summarizes the end results of various investigators. Table (3.1B) summarizes the rapidity of convergence of some of these methods. Table (3.1C) shows the rate of convergence of the circular arcs method for various starting points. Table (3.1D) gives minimization details for the circular arcs method.

Table (3.1A) End results of various methods of minimizing the function ROSIE when starting from the point (x, y) = (— 1.2, 1)

k = number of valley points found.

N = number of function evaluations

Note: The value of the function given by ref. (5) in the table is not consistent with the values given for x and y. The correct value of F equals 1(— 6) for (x, y) = (1.0001, 1.0001).

Discussion: Parentheses in column N indicate that the number of function evaluations was not published in the indicated papers, but instead

was estimated by us to correspond to each individual method, assuming an average number of 8 function evaluations for minimizing the function in a given direction. The method of row 5, ref. (7), requires the programming of the analytic expression of the gradient of F; none of the other methods do. The numbers in parentheses in the (min F) column give exponents of the powers-of-ten factors. It is obvious that the method of ref. (10) is the least efficient, while the circular arcs method is the most efficient for the function ROSIE.

Table (3.1A)

k	N	min F	x	y	Method
17	146	6(— 8)	0.999957	0.999938	"Lin. Search", this paper, 1959
—	200	2(— 5)	0.995	0.991	Rosenbrock, ref. (4), 1960
33	(264)	8(— 9)	1.0001	1.0001	Powell, ref. (5), 1962
85	(765)	6(— 5)	1.000653	1.002077	Baer, ref. (10), 1962
18	(144)	1(— 8)	?	?	Flethcer, ref. (7), 1963
12	103	3(— 9)	1.000008	1.000010	"Cir. Arcs", this paper, 1964

Table (3.1B)

	References:				
	(E) Ref. (5)	(G) Ref. (7)	"Linear Search"	"Circ. Arcs"	
k					k
0	2.4+1	2.4+1	2.4+1	2.4+1	0
3	3.6 0	3.7 0	3.5 0	3.5 0	3
6	2.9 0	1.6 0	1.9 0	7.3—2	6
9	2.2 0	7.5—1	8.3—1	8.9—3	9
12	1.4 0	2.0—1	3.4—1	3.4—9	12
15	8.3—1	1.2—2	1.1—1	—	15
18	4.3—1	1 —8	6.0—8	—	18
21	1.8—1	—	—	—	21
24	5.2—2	—	—	—	24
27	4 —3	—	—	—	27
30	5 —5	—	—	—	30
33	8 —9	—	—	—	33

Table (3.1B) Rate of minimization of function ROSIE for four different methods, all starting from (— 1.2, 1). All function values are in the powers-of-ten notation, e.g. 2.4 + 1 = 24 or 3.4 — 9 = 0.0000000034.

k = number of valley points found

The table clearly shows the better rate of convergence of the circular arcs method for the function ROSIE.

Table (3.1C) Rate of minimization of function ROSIE with the circular arcs method for five different starting points (x, y). Bottom row gives the total number, N, of function evaluations. The location of the minimum is at x = 1 and y = 1.

Table (3.1C)

x	—1.200	5.621	—0.221	—2.547	—2.000	x
y	1.000	—3.635	0.639	1.489	—2.000	y
k						k
0	2.4+1	1.2+5	3.6+1	2.5+3	3.6+3	0
1	4.1 0	9.8+2	2.0 0	5.4 0	2.2+2	1
2	3.8 0	2.6+1	8.9—1	3.2 0	2.6 0	2
3	3.5 0	1.5 0	5.5—1	1.4 0	1.4 0	3
4	3.3 0	2.4+2	2.7—1	8.6—1	2.1—2	4
5	3.2 0	1.1—4	1.4—1	2.2—1	5.2—2	5
6	7.3—2	7.5—1	5.8—2	1.9—1	1.7—3	6
7	3.3—3	4.7—1	3.5—2	1.8—1	1.6—3	7
8	8.4—3	5.3+2	1.2—2	7.4—2	1.4—4	8
9	8.9—3	4.7—1	2.6—3	1.9—2	7.5—6	9
10	7.0—7	4.9—1	4.1—4	5.1—3	6.1—8	10
11	4.4—6	3.2—2	1.2—5	2.7—3	3.7—11	11
12	3.4—9	4.2—3	2.6—7	6.4—5	—	12
13	—	2.6—5	6.6—10	3.4—6	—	13
14	—	1.3—7	—	4.3—8	—	14
15	—	5.3—10	—	2.1—11	—	15
16	—	8.9—14	—	—	—	16
N	103	195	93	118	101	N

Table (3.1D)

k	N	F	t	x	y	R	grad | total correction	
0	1	24.2		—1.2000	1.0000			
1	9	4.1	1—2	—1.0298	1.0694		2—1	
2	17	3.8	1—2	—0.9464	0.9043		2—2	
3	25	3.5	1—2	—0.8800	0.7829		2—2	
4	33	3.3	1—2	—0.8266	0.6924		1—2	
5	41	3.2	1—2	—0.7852	0.6256		2—2	
6	59	7—2	1—2	1.2705	1.6151	4.1	5—0	2—0
7	69	3—3	1—2	0.9427	0.8885	1.2	3—1	8—1
8	75	8—3	4—3	1.0918	1.1923	1.3	5—2	3—1
9	82	9—3	4—3	1.0940	1.1972	1.4	3—2	3—1
10	89	7—7	3—3	1.0008	1.0017	7.1	4—4	1—1
11	95	4—6	3—6	1.0021	1.0042	5.8	6—6	3—3
12	103	3—9	7—7	1.000008	1.000010			2—3

Table (3.1D) Details of minimization progress with function ROSIE using circular arcs method and starting at (— 1.2, 1).

k = number of valley point

N = number of function evaluations

F = function value:
for k = 0 . . . 5 in decimal notation,
for k = 6 . . . 12 in powers-of-ten notation,
e.g. "7 — 2" = 0.07.

t = accuracy achieved by F(k),
= tolerance to be satisfied by F(k + 1).

The t-values are calculated from $(F(k) - F(m))/(F(m) + C)$, where $F(m)$ is the lowest available valley point prior to $F(k)$, and where C is a suitably chosen constant (here $C = 1{,}23456$) to avoid the possibility of the divisor tending to zero. These t-values are then used as a tolerance when making the next gradient correction to find $F(k + 1)$. This automatically tightens the tolerance as the minimum is approached.

x, y= coordinates of starting points and all valley points.

R= radius of circle fitted to lowest three or four valley points.

grad correction= distance from predicted point on circular arc to gradient corrected point in valley.

total correction= distance from previous lowest valley point to new valley point.

The first five valley points ($k = 1 \ldots 5$) were found by the starting procedure described in (2.2). The first circle (with radius $R = 4.1$) is a circle through points 3, 4, and 5; its predicted low point was at $x = 6.4$ and $y = 12.2$. It was corrected to give the indicated valley point ($k = 6$). The next valley point ($k = 7$) was obtained from a circle fitted to points 3, 4, 5, 6. Likewise ($k = 8$) was obtained from 4, 5, 6, 7; and ($k = 9$) from 5, 6, 7, 8. Valley point ($k = 10$) was obtained from a circle which was no longer fitted to any of the five starting points, but instead only to the much more accurate points 6, 7, 8, 9; hence, we had a sudden speed-up in the convergence from point 9 to point 10, as evidenced also by the sudden decrease in the ratio of the grad correction to the total correction. Also, the radius R began to approximate more accurately the radius of the osculating circle ($R = 5.59$) of the valley $y = x^2$ at the minimum point (1, 1). Also, the program had automatically switched at this time from the regression plane to the regression line when predicting point 10 on the circular arc (sec. 2.5). The last valley point ($k = 12$) was the result of applying the tail correction (2.6); note the sizeable increase in accuracy.

(3.2) SHALOW

Minimum ($F = 0$) at (1, 1) with valleys along $y = x^2$ and $x = 1$. SHALOW is similar to the function ROSIE, but has a shallow valley compared with the steep valley of ROSIE.

Table (3.2)

x	—2.000	1.184	0.803	0.211	0.820	x
y	—2.000	0.574	—0.251	3.505	4.690	y
k						k
0	4.5+1	7.2—1	8.4—1	1.3+1	1.6+1	0
1	3.0 0	2.1—2	3.3—1	1.1—1	9.5—1	1
2	2.2—1	3.6—6	1.1—1	2.7—1	4.0—1	2
3	5.9—2	4.3—3	4.7—2	4.6—2	1.0—1	3
4	9.0—2	1.1—3	1.3—2	5.6—2	2.1—3	4
5	1.6—2	6.1—4	3.0—3	1.6—2	3.3—2	5
6	4.9—4	7.7—6	9.9—5	2.8—4	3.4—4	6
7	4.6—4	8.1—6	1.1—4	1.9—4	4.4—4	7
8	3.3—4	5.2—8	9.9—6	1.7—4	2.4—5	8
9	4.5—6	2.7—10	6.0—8	2.5—8	5.8—5	9
10	7.1—6	4.7—13	1.1—10	1.2—7	6.7—9	10
11	5.4—7	—	2.8—16	8.4—13	1.1—10	11
N	93	92	86	94	96	N

Table (3.2) Rate of minimization of function SHALOW with the circular arcs method for five different starting points (x, y). Bottom row gives the total number N, of function evaluations.

(3.3) STRAIT

Minimum (F = 0) at (1, 1) with a steep valley along x = 1.

Table (3.3

x	2.000	2.000	2.019	1.992	1.986	x
y	—2.000	—2.322	—1.505	—3.222	5.227	y
k						k
0	1.4+2	1.4+2	1.3+2	1.5+2	9.9+1	0
1	8.4 0	1.0+1	5.8 0	1.7+1	1.7+1	1
2	3.4 0	4.6 0	1.8 0	9.0 0	1.0+1	2
3	1.3 0	2.1 0	3.9—1	5.4 0	7.3 0	3
4	3.0—1	7.3—1	1.6—3	2.9 0	4.8 0	4
5	1.7—2	1.9—1	1.6—1	1.7 0	3.5 0	5
6	5.5—8	6.2—6	9.0—11	1.6—3	3.7—2	6
7	3.4—5	4.6—3	8.9—9	1.9—1	3.8—1	7
8	9.6—9	3.6—6	—	2.2—4	1.2—2	8
9	3.9—5	1.8—8	—	1.9—5	5.1—5	9
10	0.0 0	2.7—14	—	1.6—12	2.7—5	10
11	—	6.4—15	—	0.0 0	5.7—8	11
12	—	—	—	—	2.5—11	12
13	—	—	—	—	0.0 0	13
N	78	83	58	83	95	N

Table (3.3) Rate of minimization of function STRAIT with the circular arcs method for five different starting points (x, y). Bottom row gives the total number, N, of function evaluations.

(3.4) CUBE

Minimum (F = 0) at (1, 1) with a steep valley along $y = x^3$.

Table (3.4)

x	1.200	1.391	1.243	0.284	—1.200	x
y	—2.000	—2.606	—1.974	—3.082	—1.000	y
k						k
0	1.4+3	2.8+3	1.5+3	9.6+2	5.8+1	0
1	4.6 0	1.7+2	8.2+1	2.1 0	4.1 0	1
2	4.8—1	1.8 0	1.2 0	5.4 0	3.8 0	2
3	5.2 0	2.4—3	1.2—1	1.0+1	3.6 0	3
4	1.8—1	3.5—1	7.2—1	1.7—1	3.4 0	4
5	1.3—3	4.8—3	2.3—2	3.8—1	3.3 0	5
6	1.9—2	1.3—5	4.5—2	1.5—1	1.5+2	6
7	2.4—2	1.6—5	3.7—2	6.0—2	3.4 0	7
8	6.3—4	8.5—6	2.5—2	7.4—2	6.7—4	8
9	1.2—6	2.5—9	1.7—2	1.0—1	4.9—3	9
10	6.7—7	8.8—11	2.0—3	7.9—2	1.9—3	10
11	7.5—10	—	5.5—4	1.6—2	6.2—5	11
12	—	—	5.9—5	6.4—3	2.6—6	12
13	—	—	1.8—6	2.4—3	1.9—8	13
14	—	—	3.6—8	2.5—4	4.2—11	14
15	—	—	5.6—11	2.3—5	—	15
16	—	—	—	8.5—7	—	16
17	—	—	—	7.2—9	—	17
18	—	—	—	4.7—12	—	18
N	99	98	113	135	162	N

Table (3.4) Rate of minimization of function CUBE with the circular arcs method for five different starting points (x, y). Bottom row gives the total number, N, of function evaluations.

(3.5) DOUBLE

One minimum (F = 0.099899899) at (x = 1.001005; y = 0), another minimum (F = 0) at (x = 2; y = 0); with one steep valley along r = 1,and another steep valley along r = 2; saddle points around (—2, 0) and (—1, 0).

Table (3.5)

x	—4.000	—3.709	—3.935	—5.392	—7.633	x
y	0.000	—0.957	0.042	—1.708	—3.785	y
k						k
0	3.6+3	2.7+3	3.2+3	2.9+4	2.4+5	0
1	1.6 0	1.5 0	1.6 0	1.5 0	1.4 0	1
2	7.3—1	6.5—1	7.3—1	5.8—1	1.7 0	2
3	5.1—1	1.1 0	7.3—1	7.2—1	2.5 0	3
4	1.8—1	1.5—1	1.8—1	2.2—5	1.0 0	4
5	7.5—1	4.4—1	6.2—1	7.0—2	2.4 0	5
6	2.4—1	1.6—1	2.2—1	4.4—4	1.0 0	6
7	2.6—1	1.5—1	2.3—1	5.2—4	1.1 0	7
8	1.3—1	1.2—1	1.8—1	2.9—5	1.4—1	8
9	1.1—1	1.3—1	1.1—1	1.8—6	1.1—1	9
10	1.0—1	1.0—1	1.0—1	1.5—6	1.2—8	10
11	1.0—1	1.0—1	1.0—1	1.2—6	5.7—6	11
12	1.0—1	1.0—1	1.0—1	—	2.3—6	12
13	1.0—1	1.0—1	1.0—1	—	1.2—8	13
14	1.0—1	—	1.0—1	—	—14	14
15	—	—	1.0—1	—	—	15
N	115	113	118	106	139	N

Table (3.5) Rate of minimization of function DOUBLE with the circular arcs method for five different starting points (x, y). Bottom row gives the total number, N, of function evaluations.

(3.6) HEART

Four minima (all with F = 0) at left (0, 0), at right (20, 0), at top (10, +12.72), and at bottom (10, —12.72). Three valleys: along the x-axis, along x = 10, and along the cardioid r = 10 (1 + cos θ).

Table (3.6)

x	—1.000	—9.870	—11.43	—6.255	—7.539	x
y	0.000	—2.159	—7.344	—21.13	—27.04	y
k						
0	1.0+3	7.3+5	1.1+7	3.1 +7	6.9 +7	0
1	9.5+1	8.8+2	4.1+2	5.3 +1	8.6 +2	1
2	9.9+2	5.2+2	7.6+2	1.0 +3	1.4 +3	2
3	4.8+2	3.1+2	4.2+2	1.9 +1	6.2 +1	3
4	1.6+1	1.9+2	1.2+2	3.9 +2	7.4 +2	4
5	8.3—2	1.1+2	5.8—2	2.5 +2	3.6 +0	5
6	1.8—2	1.4+1	3.4 0	3.0 +1	3.6 +0	6
7	1.4—4	2.2 0	1.8—3	4.5 +0	1.8 —1	7
8	1.3—3	2.3+1	2.2—3	2.4 —8	4.3 —6	8
9	9.1—4	8.7 0	3.7—5	1.2 —2	2.7 —3	9
10	3.1—4	6.2 0	3.5—9	3.3 —10	4.1 —3	10
11	1.7—4	2.1 0	6.7—7	6.9 —4	4.1 —4	11
12	7.7—5	5.6—3	8.2—12	5.7 —11	3.3 —5	12
13	4.8—4	8.3—6	(bottom)	(right)	6.8 —6	13
14	9.7—5	1.7—7	—	—	2.0 —6	14
15	7.3—5	1.7—7	—	—	(bottom)	15
16	7.1—5 (top)	(left)	—	—	—	16
N	149	127	119	121	150	N

Table (3.6) Rate of minimization of function HEART with the circular arcs method for five different starting points (x, y). Bottom row gives the total number, N, of function evaluations.

CONCLUSION

We feel that the foregoing results conclusively show the gain in efficiency obtainable for a direct minimum search procedure when an attempt is made to approximate by suitable curves the valleys of the function to be minimized. The circular arcs method which strives to do so can only be a beginning. Various steps and details of its procedure can and will doubtlessly be improved in the future; most interesting should be the results of its generalization to n variables. It is possible that the gain in efficiency is larger when the circular arcs method is applied to functions with steep-sided valleys than when it is applied to functions with more clearly defined minimum points and less clearly defined valleys. On the other hand, when penalty terms are added to an objective function, in order to also observe inequality constraints, functions with steep-sided valleys will automatically be generated; thus, the methed may become useful in the field of non-linear programming. The authors wish to acknowledge the constant support and encouragement they received from Dr. Wm. J. Schart, Chief of Mathematical Analysis at General Dynamics/Astronautics, and Mr. Carl E. Diesen, Manager of Scientific Programming and Analysis at General Dynamics/Astronautics. We also wish to thank Mrs. Marilee Culver for the efficient and neat typing of the manuscript.

LITERATURE REFERENCES

1. BROWN, R. R.—"A Generalized Computer Procedure for the Design of Optimum Systems." Presented at the Winter General Meeting of the American Institute of Electrical Engineers, New York, N.Y., Feb. 2-7, 1958.
2. JOHNSON, S. M.—"Best Exploration for Maximum is Fibonaccian." RAND Corporation, Santa Monica, Calif., Report No. P-856 (1959).
3. GELFAND, I. M., and TSETLIN, M. L.—"The Principle of Non-local Search in Automatic Optimization Systems." Soviet Physics—Doklady, Vol. 6, No. 3, Sept. 1961, pp. 192-194.
4. ROSENBROCK, H. H.—"An Automatic Method for Finding the Greatest or Least Value of a Function." Computer Journal, October 1960, Vol. 3, pp. 175-184.
5. POWELL, M. J. D.—"An Iterative Method for Finding Stationary Values of a Function of Several Variables." Computer Journal, July 1962, Vol. 5, No. 2, pp. 147-151.
6. FORSYTHE, G. E., and MOTZKIN, T. S.—"Acceleration of the Optimum Gradient Method, Preliminary Report" (Abstract). Bull. Amer. Math, Soc., Vol. 57, 1951, pp. 304–305.
7. FLETCHER, R., and POWELL, M. J. D.—"A Rapidly Convergent Descent Method for Minimization." Computer Journal, July 1963, Vol. 6, No. 2, pp. 163-168.
8. DAVIDSON, W. C.—"Variable Metric Method for Minimization." A. E. C. Research and Development Report, ANL-5990 (Rev.).
9. SHAH, B. V., BUEHLER, R. J., and KEMPTHORNE, O.—"The Method of Parallel Tangents (Partan) for Finding an Optimum." Office of Naval Research, Report NR-042-207 (No. 2).
10. BAER, R. M.—"Note on an Extremum Locating Algorithm." Computer Journal, Oct. 1962, Vol. 5, No. 3, p. 193.
11. KROLAK, P., and COOPER, L.—"An Extension of Fibonaccian Search to Several Variables." Communications of the A.C.M., Oct. 1963, Vol. 6, No. 10, pp. 639-641.

ON THE EVALUATION OF THE COST-EFFECTIVENESS OF COMMAND AND CONTROL SYSTEMS

N. P. Edwards
Weapons Systems Evaluation Group
Washington, D. C.

INTRODUCTION

1. some time now I have been previleged to be associated with a project which has been taking a critical, and hopefully, constructive look at the workings of the National Military Command system. Working in the Pentagon one regularly hears the expression "Cost-effectiveness." Sooner or later someone had to combine command and control with cost-effectiveness. This paper then addresses the cost-effectiveness of military command and control systems.

2. Command and control is a broad and very active field as can be seen from the many articles in military and trade publications and in the popular press. For instance, the *Armed Forces Management* magazine devoted its July 1963 issue to command and control. I would like to review some of the titles of the various articles:

—"NMCS: The Command Backup to Counterforce"
—"Total Command Control Through Computers?"
—"Taylor's Flexible Response Strategy Shifts C&C Perspective"
—"Electronic C&C Will Aid Army Commander, Not Replace Him"
—"We Have More Than 800 C&C Systems"
—"Resource Management: A New Slant on C&C"
—"Real Time C&C Aids USAF, NASA"

One of these also states ". . . C&C system is a communications system . . ."[1] Under the aegis of C&C, *Electronic News* reports that Mitre has developed ". . . a new approach, whereby computers will be provided with a broad, flexible program which contains nothing regarding the using commanders' specific problems. The program—which will be a 'learning program' based on artificial intelligence—will enable the computer to learn by experience as the system commander feeds his specific problems into it."[2]

3. From this small sample of the current literature, it can be seen that the expression "military command and control systems" has a wide range of meaning to different users, and covers a broad scope of systems and activities. If a meaningful attempt is to be made to discuss cost-effectiveness of command and control systems, it is essential to find out what we mean by command and control. So let us begin by defining the individual terms and attempting to break the problem of cost-effectiveness into manageable segments for examination.

4. "Command and control" as defined in the *Dictionary of the United States Military Terms for Joint Usage* (*JCS Pub. 1*) is "An arrangement of personnel, facilities and the means for information acquisition, processing and dissemination employed by a commander in planning, directing and controlling operations." Note that the word "system" is not used.

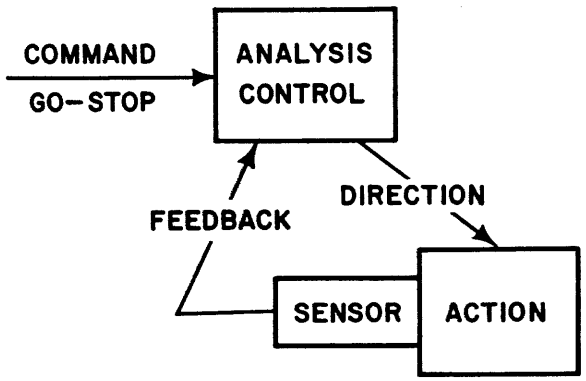
5. A "command and control system" as the term is normally used, includes many different

functions. In order to begin to seperate these so that they can be handled individually, let us differentiate between the command function and the control function. We will first define and describe the characteristics of control systems: These are a class of systems sometimes included in the general definition of command and control, but which we propose to exclude from the subsequent discussion, except for use as a basis for analyzing the command subsystems we are really interested in.

CONTROL SYSTEMS

6. As normally used in the technical sense, a control system regulates the action or activity of a person or device in order to perform a prescribed function. The key point here is that the mission to be accomplished is understood exactly and the means of accomplishing this end are also well understood. A control mechanism operates in accordance with precisely defined laws. It senses the performance of the controlled device and corrects deviations from the desired course in accordance with mathematical formulae. Control systems frequently have people in the control loop. An aircraft controller is an example of a human link in a control system. While the controller does not operate in a completely predictable mode, the majority of his actions are predictable in any except an emergency situation. In a weapons control system, of which SAGE is probably the largest single example, the majority of the functions can be described mathematically. Those functions which cannot be described mathematically are referred to human judgment. However, the ultimate goal is to eliminate the requirement for human judgment from the control system once the command to execute has been issued. Some systems, such as missile weapons systems and antimissile systems, operate at such a high speed that it is necessary to eliminate the human operators from the control system loop. A distinguishing characteristic of control systems (as defined in common technical usage) is "feedback." This is true regardless of whether the system controls things or people. "Feedback" is the process by which a control system constantly senses what is occurring and adjusts the system to achieve the desired result. An aircraft controller directing the landing approach of an airplane uses feedback. He continually follows the actions of the plane and gives correcting instructions in case the plane is not following the desired course.

7. The diagram in Figure 1 is an oversimplification but it is intended to illustrate the dependence on sensors and feedback, and the prescribed nature of the action.



CONTROL SUBSYSTEM

Figure 1.

COST AND EFFECTIVENESS OF CONTROL SYSTEMS

8. Various control systems which are completely (or for the most part) mechanized, have been in production for many years. Such systems include aircraft autopilots, instrument landing systems, missile guidance systems, tank and naval gun stabilizing and aiming systems, and many industrial process control systems. In the case of an industrial control system, such as a system controlling the operation of an oil refinery, cost and effectiveness trade-offs have been very carefully examined and are well understood. This is possible because the processes, procedures and desired end products are well defined. This is also the case in many small military control systems. Although the value of increments of improvement in the performance of a military control system may be difficult or impossible to measure, the value of an improvement in the performance of a public utility generating plant can be measured with great precision. An increase in operating efficiency of 0.1% may represent a significant increase in the profit margin.

9. There are three factors which predominate in their effect upon the accuracy of a cost-effectiveness estimate for a new control system. These factors are: value of performance, how well the function to be performed has been defined and the performance level desired.

a. Value of Performance. Can a value be assigned to an incremental improvement in the performance of the system? In many instances it is possible to identify a significant improvement in performance and assign a value to it. Two performance factors of interest in the control subsystem of a weapon system are:

(1) *Reliability*—If the reliability of the weapons control system can be improved, this improvement can be reflected in the overall reliability of the weapon. If the control is the least reliable subsystem, then the improvement will probably be quite significant. If the control subsystem is already the most reliable part of the system, then the improved reliability may have little effect on the weapon's performance. However, it might still be well worthwhile because of a reduced field maintenance problem. The total system cost and increased efficiency must be taken into account.

(2) *Accuracy*—An improvement in the accuracy of the control system may significantly reduce the number of weapons required if the target location is known to a higher accuracy than the accuracy of the weapon. Suppose the target location is known "exactly." If the accuracy of delivery of the weapon is such that the target will always be within the radius of destruction, only one weapon (shell, iron bomb, or other) is required. If the error in aim and delivery is greater than the kill radius, we have a more expensive and probably less efficient situation (see Figure 2). The number of weapons required goes up logarithmically as the probable dispersion gets significantly larger than the radius of destruction of the weapon.

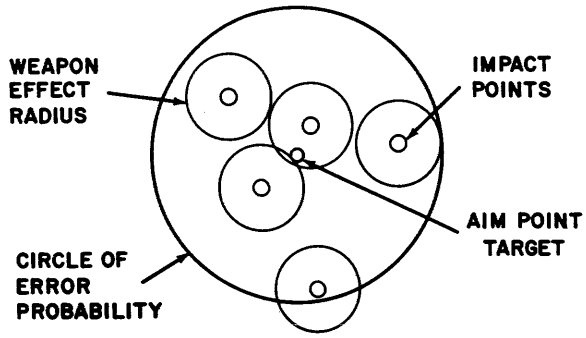


Figure 2.

Formulae and nomographs exist for calculating the weapons required for a desired probability of destruction of the target.[3]

If you cannot assign a meaningful value to incremental improvements in the system, a cost-effectiveness projection cannot be made. However, analysis can still be used, and it still may be possible to make a relatively accurate comparison of the projected cost of different approaches and rely on experienced judgment to make a proper (or best possible) decision using the facts made available by careful analysis.

b. Definition of Function. Are the functions to be performed by the control system defined and will they remain constant during the design, test, and production of the system? If so, one of the necessary conditions for an accurate cost projection has been met. If not, it is certain that the initial cost estimates will be significantly in error. The more poorly defined are the functions to be finally performed, the greater the probable error in the cost estimate. It would appear that it should be relatively easy to determine the functions required of a control system. However, if the device to be controlled is being developed concurrently, and by a different engineering group or company, as is frequently the case, it is probable that control requirements will occur which were not anticipated. It is impossible to describe this risk numerically in the abstract, as each case will be different.

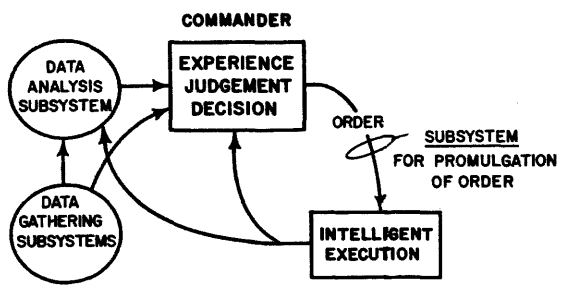
c. Performance Levels. Is the system's performance *specified?* Will this performance re-

main essentially *constant* during the design, test, and production of the systems? If so, another necessary condition for a good cost estimate has been met. If not, the estimate has a high probability of being wrong. Is the performance required comfortably within the present state of art? If so, a major condition necessary to permit an accurate cost estimate has been met. If not, R&D is required and the estimate may be in error by factors of tens or hundreds.

10. Even where a new hardware/software system is of a kind which has been designed and produced repeatedly, accurate cost-effectiveness measurements or predictions cannot be made unless the value of what is to be accomplished is known, how it will be accomplished, and whether it can be done without R&D. This discussion has been included to illustrate the minimum conditions which must be met before reasonable cost-effectiveness projections can be made of any system or equipment.

COST-EFFECTIVENESS OF "COMMAND AND CONTROL" SYSTEMS

11. For this discussion, three parts of the "command and control" function will be considered. These are more accurately called "Command and Support Systems." Figure 3 is a very simplified version of a very complex and personal activity. It is complex because it is an activity of sophisticated people in a complicated and changing environment, reacting to ambiguous situations which are inaccurately and incompletely reported. It is personal because each



COMMAND SUBSYSTEM

Figure 3.

commander has his own particular likes, dislikes, prejudices and means of operating. The commander will augment any formal "subsystem" we may build in any way he sees fit. The augmentation may be temporary or permanent (for the tenure of the commander) but it is not part of the subsystem discussed here. Realizing that we are discussing command support subsystems, let us consider each of the three selected which are:

a. Data-Gathering Subsystem. A subsystem which gathers information necessary to support a command decision.

b. Analysis Subsystem. A subsystem which provides or supports the analysis of the information. (The command function is not considered a subsystem.)

c. Transmission Subsystem. A subsystem for the promulgation of the decision to the appropriate subordinates.

12. The function of command, i.e., evaluation of the information and the command decision, is not addressed. This simplified diagram would emphasize the facts that:

a. Commanders use many inputs, including feedback.

b. The commander knows and relies on the intelligence and experience of the person responsible for executing the order.

13. Webster's New World Dictionary uses the phrase "Command . . . when it refers to giving orders, implies the formal exercise of absolute authority . . . order sometimes suggesting an arbitrary exercise of authority . . ." This is a clear recognition of the fact that it is not possible to describe rigorously the transfer function involved in the command process. This is true not only in general; it is difficult to think of a single specific command situation in which a series of logical conditions can be defined which describe the specific performance required of the commander. However, it is possible to describe a few selected situations in which the correct performance can be readily predicted and completely defined. A doctrinal response is such a case. However, it is easier to describe one of the infinite number of situations in which the correct performance

cannot be completely defined in advance. Also, it will probably not be possible to determine if the decision was correct for days, or even months or years.

14. Since the command process itself cannot be described precisely, or for that matter with any degree of precision whatever, it is meaningless to talk about cost-effectiveness predictions for the command function. Specific arbitrary situations can be defined (as is done in a war game), decisions postulated and the predicted results evaluated. This is useful to contribute to the learning process of commanders. It may also be useful to assist in evaluating a commander's thinking in the absence of a real war (certainly one would not want to leave the fate of a real military decision exclusively in the hands of a computer war game). While industry has for some years used a business game as a means of training and evaluating executive management, they do not leave real business decisions in the hands of this business game.

DATA-GATHERING SUBSYSTEM

15. Any commander (or business executive) needs information on which to base his decision. Military data-gathering systems have existed since before man first picked up a club. General Custer had a data-gathering system which provided him with adequate information to have prevented the massacre. His scouts told him that the Sioux were as numerous as blades of grass on the prairie. It was in the data evaluation that Custer's command system was at fault.

16. Since some data-gathering systems are well understood, it should be possible to establish a cost-effectiveness comparison between various systems designed to achieve a specific performance. However, remember that this comparison is subject to all the conditions which limit the accuracy of cost-effectiveness estimates described above for *control* systems. It should also be possible to predict (with varying accuracy) the cost and performance of a given data-gathering system. However, it is certainly not always possible to accurately assess the value of increments of performance of a data-gathering system. No generalized observations can be made concerning timeliness, accuracy or reliability of the system. These are functions of the specific situation. Consider these major performance factors:

a. Timeliness. How much is it worth to have the data a day, hour, five minutes, or ten seconds sooner? This is completely dependent upon the nature of the system, the nature of the situation, and the nature of the data. Given a specific data requirement, it is probably possible for an experienced military commander to put an arbitrary (approximate) value on the timeliness of the data. It is the commander's responsibility to make this decision since he is the one person who is held responsible for the consequences of his decisions.

b. Accuracy. How much is accuracy worth in a data-collection system? This again is dependent upon all the factors listed above and also on the accuracy of the raw data and the quantity of the data. Given a specific requirement for data, arbitrary and approximate values can be assigned by the commander. It is not possible to do this in the abstract. (The accuracy of the system could be defined as the percentage of the data entered into the system which arrives unchanged at the output of the data-collection system.)

c. Reliability. This could be defined as the percentage of the time that the system is performing in its normal manner. What effect will a temporary or permanent failure have? Here again, the final answers are dependent on the specific system and can be assessed by experienced judgment only.

17. To reiterate, it is possible to compare the relative cost and performance of various data-collection systems designed to do a specific job. If the performance is equal, or assumed to be equal, the relative cost can be compared. The problem is, proposed systems are likely to be only approximately equal and then only where a system has been designed in response to a rigid set of specifications such as may be the case when contractors respond to a request for proposal. Even here a data-collection system is usually rather complex and the variations of system performance implicit in the proposals

are very likely to be of significance to the military commander.

18. Certain types of command situations permit a relatively accurate and profitable assessment of the value of timeliness, accuracy and reliability. Consider the case of a moving target with a known top speed. Knowledge of its exact present location is limited by the speed, accuracy and reliability of the reporting system. If we don't know of any restraints on its direction of travel, we must assume the target has a certain probability of being within a circle whose radius is determined by its speed and the age and quality of our knowledge of its last position. If we assume, for simplicity, that we have an accurate, reliable delivery system and a certain radius of kill, we can calculate the number of weapons which must be applied to the target area to give a desired probability of destroying the target. A method giving minimum overlap of destructive areas is the inscribed hexagon model.[4] The target area is covered by hexagons inscribed into radius of damage circles. If the radius of the target and the radius of desired damage are equal, one weapon is required. If the radius of the target is 5 times the radius of destruction, 37 weapons are required. (See Figure 4.) The number of weapons goes up as a function greater than, but asymptotic to, the square of the linear uncertainty as to the location of the target. This uncertainty includes, when you are estimating the number of weapons to stock:

a. Reporting accuracy.

b. (Speed of target) x (Probable reporting time loss).

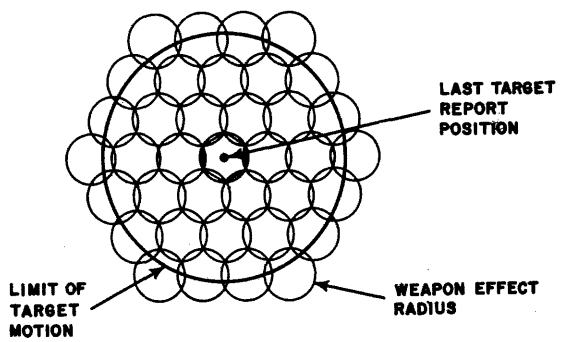


Figure 4.

c. A safety factor to allow for the fact that the information you have may be older than you think (*reliability* of the reporting system).

This analysis still will not give you the total cost of being effective, because it does not include such things as the probability of being able to deliver more than one weapon, time loss if weapons are delivered sequentially, or the reaction of the target if you miss the first time and get it excited, etc., etc. This simple example is virtually a control problem, but illustrates the point.

19. Assignment of absolute values to the total aspects of the performance of these systems requires the highest knowledge of the situation, the military strategy, policy and innumerable other factors relating to the national welfare. Hence, it is necessary that judgment be exercised in making the decision as to which system is selected. This judgment can never be made with full possession of all pertinent facts. It will always be subject to human error and can always be challenged. The military commander is held responsible for the consequences of this decision and, therefore, must have final responsiblity for making the decision. This does not belittle the contributions that can be made by proper application of analytical techniques. Careful analyses and cost projections of competitive systems can highlight areas of uncertainty as well as provide evaluations of known facts, and should greatly enhance the probability of the commander making the correct decision.

ANALYSIS

20. While the function of analysis has occurred in the commander's mind, with the aid of his staff, since the beginning of military action, it is possible to describe precisely only very limited aspects of the total field of information analysis for military command. In spite of this, much of the analysis requires the rapid and orderly retrieval of data from a large body of reference and current operational information. Facts relating to force status, logistics of supply and movement, geography, climate and installations are required, to name some of the more important. These must be stored in an

orderly fashion. However, to support analysis, the facts must be related and sorted into many different orders or categories. Various manual or mechanized systems to aid this activity can be compared for cost, convenience, speed and other factors. Statements can be made regarding cost and relative performance. In some cases it may be possible to put a specific value on a difference in performance, but usually it requires command experience and judgment to make the decision. And the decision as to the worth of a particular fact, array of facts in a category, or the worth of specific analysis may change drastically when the commander changes. Here, perhaps more than in any other command support subsystem, the effectiveness can be assessed only in the mind of the commander. What does *he* think he needs?

SYSTEM FOR THE PROMULGATION OF ORDERS

21. Here again we have a function which has been performed by the military since the beginning of organized warfare. Given a specific situation, or a general requirement, it is possible to describe many different approaches to disseminating command information. These have included word of mouth, runner, carrier pigeons, bugles, drums, flags and rockets. Again, it is possible to compare the relative cost of various systems only if they have equal performance. Again, it is a matter of military judgment as to the value of improved performance in such a system. Accuracy is extremely important. Reliability, i.e., probability that the command will be delivered, is of great value. The value of speed may be dependent in part upon the response time of the force commanded. A command transmission which takes seconds may be of no greater value than one which takes minutes where the response time of a military unit is a week. However, minutes or hours may be important even though the response time of a unit is days or weeks. It is possible to miss even a slow boat. To establish values of various system performance factors usually requires military judgment and cannot be generalized. Each planned application can be studied and in many cases definite, but not definitive, values can be attached to improvements in performance of some part of the system. Other factors which cannot be assigned specific values, such as giving the order a greater sense of urgency or authenticity, may be more important.

22. It is worthwhile to single out the specific case of a nuclear exchange. This is certainly the most thoroughly studied of all the possible military activities which are not yet a matter of history. The nuclear exchange, once it is initiated, is usually considered to be more nearly a control problem than a straight command problem. It is bounded by doctrine, a finite number of weapons and target choices. Command support subsystems with specific tasks can be evaluated for specific cost and relative effectiveness. A missile detection system can be evaluated—with existing missiles—and its performance against hypothetical new missiles estimated. An analyst can even assess many of the advantages of having a few minutes more warning time. But an evaluation of the dollar worth of this time is a task for the top military policy makers. Values can be assigned to degrees of reliability and accuracy. These can contribute to a sound decision, as they would in any other weapon of destruction system. In general these are like or are really control subsystem problems and can be addressed by an analyst with considerable hope of obtaining some meaningful results.

23. The command problem in the transattack and postattack period is more complex and probably less understood than conventional warfare. We don't have any relevant experience since fortunately we have never had an exchange. There are several questions we can ask about any proposed subsystem for command support:

a. Can the system survive to do its job?
b. Will there be anyone to use it?
c. Will there be any data to feed it?
d. Will it help?
e. What will it cost?

24. Prior to this point an assumption has been made that hasn't been stated. It has been assumed that the subsystem or system component does something useful—it contributes to the better solution of the problem. This is the first question: Is this system relevant? The

answer to this may save a lot of analysis time and many procurement dollars. If the answer is "Yes, the subsystem is relevant and useful," the following general observations can be made:

a. Two or more approaches to the solution of the problem can be compared as to cost and performance factors.

b. With luck, one of the alternatives may be both better and cheaper.

c. If additional performance is offered by the more expensive alternative, the problem arises of assessing the value of the increment of performance. Even if dollar values can be assigned, there are always other less tangible operational factors which must enter into the decision.

d. A choice may exist as to whether or not to have a control system (or a particular weapon or device), but no choice is offered in the case of a command system. The command systems exist now. So we are always offered a relative choice. We can use present methods or add support subsystems. These can always be compared with the existing way of doing business. In either event we have the basic command structure. Command systems are not a question of hardware.

e. In all system component choices we may be faced with the cost estimator's standard dilemma. System A may use a part of an existing system, while System B may have growth potential or reduce the cost of future systems.

f. The analyses and quantifications which can be done using the techniques provided by the engineering and scientific communities are valuable tools which can assist the commander in making the best of an extremely difficult situation.

g. The final decision as to the cost-effectiveness of a military command and control system should rest with the commander who is responsible for using the system and for the effect that it has on the success of his mission and the national security, whether he is a component commander or the Commander-In-Chief of the U.S. Armed Forces. He should have all available help, utilizing the best analytic techniques, in order to make the best decision.

25. The commanders of the Unified and Specified Commands have been, in fact, given the responsibility for their command and control systems by the Secretary of Defense on 26 October 1963, according to *Armed Forces Management,* January, 1964. The Commanders will be looking eagerly for help in solving their extremely knotty and complex command support problems. I believe that virtually everyone here considers himself a professional, and many of you will be called upon to help solve these problems. Let us take a responsible attitude, and recommend hardware and software only when it is relevant to the problem, and can contribute to the solution better than an improved manual system, and only then after we are certain that the improvement is worth the cost; and that the user wants the kind of solution we propose. And I mean the user, the man who will have to make the system work as a part of his daily job. Let us have the courage to say, when necessary, "Glamorous hardware and elegant software alone won't solve your problem." And then work with the people who have the problem, and together find out what the problem is. And perhaps find the solution, which may even make use of good hardware and software. Our professional ethics and pride require it.

REFERENCES

1. ROBERT G. PFEFFERKORN, *"Taylor's Flexible Response Strategy Shifts C&C Perspective,"* Armed Forces Management, Vol. 9, No. 10, Page 35, July 1963.
2. RAY CONNOLLY, *"System Evaluation is Underway at ESD," Electronic News,* Vol. 8, Page 22, March 18, 1963.
3. JOSEPH S. TOMA, *USAF, "Probability Applications to Weapons Systems Analysis,"* AFSWC–TDR–62–59.
4. *Ibid.,* page 29.

FRACTIONIZATION OF THE MILITARY CONTEXT

Dr. Frederick B. Thompson
TEMPO, General Electric Company
Santa Barbara, California

INTRODUCTION

Every subject is said to have both an inside and an outside. Its outside has to do with the relationships between the subject and matters external to it; while its inside is concerned with the subject's own internal dynamics. My talk concerns the dynamics of information in today's institutions with specific reference to the problems of command and control. Thus I shall leave questions concerned with the relation of command to strategy and the nature of future wars to my colleagues who are well prepared, as I am not, to discuss them.

Throughout my remarks I shall make a great deal of use of the word "context." And so I shall first clarify just what I mean to convey by that word. I shall mean by context the total view a man, or a group of men, has of his situation at a given time. Thus context includes knowledge of environment, historic perspective, plans for action, motives, goals, all aspects of our understanding of the what, how and whys of our current situation. The context of a military headquarters is rather well-delineated. It includes the personnel records of the command, intelligence, operational information such as status of forces, logistic records and plans, operational plans, communication procedures, and certainly a clear understanding of the command's mission.

In its broader meaning, command and control are the means by which an organization maintains adequate and interconsistent contexts for its constituents. The inside problems of command and control are the problems of maintaining adequate, relevant contexts under changing conditions and problems, and of insuring consistency across the varied and changing contexts of the members of the organizational hierarchy. It is on these problems of contextual dynamics that I would like to focus your attention.

THE SELF-ACCELERATION OF CONTEXT

Our contexts tend to grow. The more we know, the greater our curiosity, the more the remaining unknowns provoke us. Each new discovery seems to create more problems than it solves, problems which beckon us to more complex, more extensive theories and understanding. This quest for knowledge is not only for facts concerning the external world, but also for a better understanding of ourselves, of our wants and objectives, and of the relative values to us of the alternatives we perceive. Context is ever-expanding.

If our contexts were limited to those which could be contained in the head of a single individual, our knowledge would be limited indeed. We have learned to build communities, communities whose principal function is to build and maintain a group context which exceeds in information that of any of its constituents by many orders of magnitude. Specialization can be viewed as a fractionization of the group context in such a way as to maximize the total information in a community that can be brought

to bear on its various concerns. And the essence of organization is to insure intraconsistency across the group context while maximizing its informativeness.

Our motivation for this insatiable quest for ever greater knowledge is clear. The more we know, the more efficiently we can modify our environment to meet our desires. Certainly our current standard of living is ample evidence of what the expanding human context can mean to us. But if expanding context brings greater control over our environment, it also extends our ability to gain knowledge itself. The great extensions of context made by Copernicus and Galileo and the observations of Tycho Brahe, opened the door for the kinematics of Kepler. In turn these advances showed the way for the dynamics of Newton, and so on through the discovery of the Van Allen belt and solar winds. This same effect is equally applicable in the small as in the large. When a boy learns to ride a bike, this new knowledge vastly increases the world he can explore. Contextual growth is self-accelerating.

Look at this fact very carefully: Context is self-accelerating. If context were limited to a single individual, there would be a ceiling on contextual growth and this acceleration would be controlled. But by building larger and larger communities, we have removed this bound on contextual growth. If the community context were to grow steadily, one could conceive of an ever-enlarging culture. But the self-acceleration does not permit steady growth. As the *rate* of growth of knowledge increases, as well as its overall extent, self-consistency of the total construct becomes strained. Communication across the context cannot keep up with the new discoveries and conceptual extensions taking place on all sides. The feeling of being out of touch rapidly deteriorates to a fractionization of community identification. The basis of communication, the commonality of context, disintegrates. Castastrophic fractionization of context ensues, and the result is the plummeting of the amount of information remaining in the community.

The symptoms of contextual fractionization are many. Among them are, for example, a feverish preoccupation with the problems of military command and control, attempts to centralize control of departmental or corporate informational activities, acceleration of organizational change, greatly exaggerated evaluation of communication activities and increase in "paperwork" of all kinds, and standardization by fiat, rather than by community recognition of common practice. And the final desperate stage to secure the shattering informational community is dictatorship.

Is such catastrophic fractionization of our communities inevitable? I do not know. Certainly the reading of history gives us pause to wonder. But there are ways of dealing with the problem of contextual explosion. Context must fractionate, because the strains on communication do become excessive, strains which build up faster than technological advance can relieve them. But we can achieve orderly fractionization. The constituent communities can be organized to retain through the organizational hierarchy the means to bring to bear a total body of know-how and understanding not achievable by any one element of the organization.

Gentlemen, the Department of Defense is tending toward catastrophic fractionization. The almost explosive rate of contextual expansion is all too clear. Intercontinental missiles are the result of global concerns and scientific advance, not vice versa. And the rate of expansion of the total defense context, indeed, of every facet of that context, has become explosive. There is no reason here to recite examples of the disintegration of communications, of inconsistencies of policy, or of the increasing difficulties of initiating definitive actions. We are all too aware, each in our limited sphere, of this fractionization. And even the newspapers and trade journals confirm our fear that it may be widespread.

Those of us gathered here represent a sizable part of the community that has both means and mission to reverse this rapid decay in overall information in our monolithic organizations. We can assist in the achievement of ordered fractionization that can in turn increase the potential for information of today's institutions, and thus our potential for decisive, efficient action. But it is not in the increase in commu-

nication, the centralized data centers or the paternal agencies established in the name of efficiency, that this reversal will be achieved. Efficiency in the informational activities of a community come from entirely different sources. Before discussing specific approaches that our industry might undertake to give effective service at this critical time, I would like to take an overview of the kinds of measures a community can take to accommodate its expanding context—measures I believe the Defense Department must take if it is to maintain an adequate defense posture for our nation.

STEPS TOWARD PREVENTING CATASTROPHIC FRACTIONIZATION

Organize to Minimize Communication Requirements

The task is not that of maintaining a single, all inclusive context where the centralized Secretary's Office has all the details of its far-flung operations at its fingertips. No single context can accommodate it. It is not just that no one man can take it all in a single thought. It is simply impossible to put such a context together no matter what resources are brought to bear. The means of communication are not adequate, nor can they be made adequate, to maintain the consistency of such a context whose rate of change in every facet is as high as it is today.

The essence of traditional command doctrine is the delegation of command prerogatives to the responsible officer on the scene. Let us examine this doctrine for a moment. What do we mean "on the scene?" In a task force operation the man on the scene is the task force commander. He is directly in touch with his subordinate command elements which he controls and maneuvers in the accomplishment of the task force objective. However, if we consider one of these subordinate units, say a combat ship or combat wing, the commander of that unit is on the scene relative to its control and the coordination of its operation. In turn, for the operation of the ship's power plant, the engine room officer is on the scene; for the operation of a particular aircraft, the aircraft commander is on the scene.

A military operation — any operation — directly deals with reality; not with hypothetical plans, nor with far-flung systems which can never be fully sensed or directly experienced, but with individual men and individual items of equipment, and individual reports and individual points of view. This reality is strangely peculiar to the particular operation. It may be an operation at the highest level, such as dealing with Rusk and McNamara, Khrushchev and De Gaulle. It may be an operation at a middle level, such as the execution of a blockade of Cuba dealing with particular ships, the personalities of well-known officers and with weather conditions you have personally tasted on many a watch. Or it may be an operation at the lowest level such as knowing instinctively the coverage pattern for a radar and the kind of returns to expect, and the sense to be able to distinguish the moon from incoming missiles. Traditional military command doctrine gives the responsible officer, who is in the position to directly sense the reality, the means and the prerogatives to deal with that reality "on the scene."

The essence of this doctrine is twofold. First, the responsible officer on the scene is in the best position to have the relevant information. Second, the delegation of prerogatives assumes responsible reporting of the relevant facts upward. As a result, the *requirements for communication are reduced to a minimum and the relevance of communication is maximized.*

To attempt to obtain at a far-away headquarters the essential details, the peculiarities of an operation, and to exercise detail control from a distance puts impossible strains on communications. I do not mean by "communication" solely the transmission and reception of signals, but all that is implied when we say two men are communicating, one to the other. Those who think that this problem can be solved by great automatic data systems—and there are many people who think this in our industry—have no appreciation for the complexities of military operations, though they would decry the difficulties of debugging a simple computer program by telephone. The result of such systems is inevitably the inundation of the higher headquarters with noise and the rapid deterioration of the effectiveness of the subordinate.

The notion is also popular that one can centralize a portion of an operational task in a central agency, far removed from the realities that give meaning to the subtask in the first place. Thus, for example, major functions of military intelligence and military communication are centralized for the sake of efficiency. But the sources of feedback and the measures of relevance germane to the operational realities and requirements are missing in such centralized agencies. The relevant measures, heard only indistinctly, are all too easily replaced by feedback from the local environment which are all too often at cross purposes with the needs of the field.

One of the control centers of a unified command wished to move a telephone from one side of an office to the other. It took two years for the paperwork to be processed through Washington before official permission was given. Of course, the phone was moved long before that. But it is precisely the inconsistency as well as communication strains which result from such foul organization and failure of delegation with which we are concerned.

An equally ridiculous example is that a major military command now is in the process of putting all their personnel records for the last ten years on punch cards and sending them to Washington. Washington may be able to process punch cards very fast, but any manager who has had to match man to job and stay on top of the personnel matters of his employees can imagine the reduction to primitive levels that managers on the scene will be forced to employ and the stratagems that will be used to keep Washington permanently misinformed.

To delegate operational tasks to men on the scene calls for attention to the kind of tasks which are likely to occur, the kinds of control which must be maintained at higher headquarters and strategic considerations. These are the outside aspects of command and control. But once these are properly assessed, the inside principle must be followed: *organize and delegate to minimize the volume and maximize the relevance of communication.* And the key to this principle *is to assign responsibility and prerogatives to the man on the scene.*

To Stay Consistent, Stay Close to Reality

One of the greatest deteriorating factors in any organization is incipient conviction on the part of a constituent that others with whom they must deal are out of touch with the realities of the situation. If the higher military headquarters appears to be functioning on the basis of assumptions that the field knows by its direct observation to be false, or even if the field only believes them to be false, the field will inevitably respond by subverting the information structure to its own ends. If the higher headquarters loses confidence in the meaningfulness of the reports it receives from the field, it will ignore those reports in making its decisions. The result is rapid collapse of communication, a collapse that cannot be detected in bit rates. The problem of maintaining touch with reality is greater now than ever before, for we have so few bench marks on which to lay our conceptions of military operations. We had not had operational experience under full scale war time conditions with weapon systems and unit organizations with which we are now equipped. There is no underlying body of experience that, among other things, assures the consistency of view that is so necessary to maintain the cohesion of the military organization. Thus we must insist with uncommon concern and tenacity that the operational realities of which we *are* cognizant are kept firmly in mind, let the chips fall where they may.

In many cases, the knowledge of our senior military officers concerning what is actually going on in their commands is simply inaccurate or misinformed. It is generally accepted that the SAGE command and control system led the way in the development of modern automated command control techniques; that SAGE was the great prototype of automated command and control. The indications are, however, that many SAGE sites are not exercising their equipments from one inspection to the next, that some have even put cellophane overlays over their scopes and returned to essentially manual operation. What confidence can we put in the development of future command and control systems when SAGE is named as the great prototype?

But it is not only that our higher headquarters are nct looking. Of much greater concern

is that we are ignoring at all levels realities all too apparent to the analyst who will but unflinchingly lay out the known facts. How many systems, organizations, bureaus and military commands live on when both they and their missions are obsolete? What is the degree of collusion in the conduct of military exercises? What studies have been buried? Who has not felt too often the pressure to justify rather than establish, and witnessed too many contrived demonsrations?

We are not looking; we are not probing below the surface. But we have found a substitute: the operations analysis that purports to "simulate" military operations in a computer or establish formulas for determining the cost effectiveness of weapon systems. Operations analysis can be useful, but only if soundly established on adequately developed experimental evidence from the field.

It is important that those who are applying the results of such analyses at the strategic level understand our technical deficiencies in the basic and crucial assumptions of these studies. A few specific examples of such deficiencies are:

- there is no adequate method of discounting over the long haul in cost/effectiveness studies;
- economic comparisons or economic resource cost estimates involving the Soviet Union depend on an adequate analysis of the ruble/dollar ratio, which has proved intractable to date;
- the "soft" effects of nuclear weapons on population, communications and organization appear to dominate the "hard" effects, yet we have no adequate means beyond intuition to project these "soft" effects;
- there is no adequate method to assess the effectiveness or the effects of deterioration of combat communications of any of the military or diplomatic services.

Yet these deficiencies lie across the very roots of every major operations analysis today.

Eight years ago I participated in a study to determine whether the results of a certain class of effectiveness studies were sensitive to the underlying parameters of the studies. We found that the degree of sensitivity, relative to our knowledge of the parameters, reduced the significance of the results to a point where their usefulness was in grave question. I recently examined this same area to see what improvements had been made. I not only found the same practices, but the same reports which I had presumably established as inadequate were quoted as the basis for current studies.

Is it not the case that our studies grow from roots that penetrate only as far as the intuition of the analyst? One of the best known and respected operations analysts once showed me his notebook where he wrote down prior to undertaking a study what results he expected to prove. He was very proud that he had never failed to predict his results correctly. If I were to be asked to choose between the expertise of a Ph.D. mathematician and a flag officer of our armed forces in matters of mathematics, I would pick the mathematician. As a mathematician, I do not have the arrogance to hold that the mathematician's "guesstimates" and intuition should hold sway in the arena of military operations.

I am not here decrying lack of objectivity as such, or pointing in alarm at malpractice. What I *am* saying is that if we are to reverse the catastrophic fractionization of context which is occurring in the Department of Defense, we must insure by all available means that consistency which can legitimately be found. The best source of consistency is reality itself. We must get out of our nonprofit ivory towers, administrative labyrinths, and self-mesmerizing brochures. We must insure that our context matches the reality that is there, if we bother to look. Experience must again be valued, the man on the scene respected, and the hunch heeded that tells us we had better drop in unexpectedly to get a first-hand look.

The Importance of Purpose

Modern communication theory teaches us that information can be measured in terms of our relative uncertainty concerning a set of alternatives. What it does *not* tell us is where these alternatives come from in the first place. Once the alternatives are clearly defined, then that part of a communication which does not

properly reduce our uncertainty is identified as noise. However, if we were to change our selection of alternatives, what was once noise may become highly informative communication. Thus the selection of alternatives is a crucial aspect of obtaining information.

However, it is in the selection of alternatives that we often go astray. It is far too easy, when we come across a new facet of a situation, to either make simplifying assumptions related to the techniques of the analysis rather than the problem, or to simply ramify the alternatives, establishing new requirements for data. In either case, communication theory gives us neither reason nor warning for the resulting dilution of relevance. We pay homage to systematic methods, and, like the physicist who ignores viscosity in favor of Newton's laws, forget about the sticky aspects. We must be sure to get all the hard facts; to distrust—indeed fear—the man who is outspokenly biased. We dare not make judgements of relevance, and the only limitations on alternatives are those made in favor of allowing more precise analysis. And as a result, we are faced with the paradox that since there is nothing we dare call noise, it is all noise. I know of one office of the Defense Department that has stored away over 7000 magnetic tapes, each containing a million and a half words. They never have and never will have the capability to look at them, yet they are reluctant to throw them away.

The selection of alternatives is where our purpose is revealed, where our measures of relevance are established. It is only when there is clear purpose that there exists the means for establishing relevance. And it is only through a sense of relevance that we can winnow out the essential elements of information from the swirling, encompassing chaff of data and reports. When we have our purpose clearly in mind, then the one or two alternatives that we actually have stand out clearly in front of us.

What is purpose? Purpose is not something set down in writing under the subhead, objectives; nor is it the first few paragraphs of a justification paper. I imagine that each of us has on occasion sat with tablet and pencil before us trying to put into a position paper or proposal a meaningful statement of purpose, and finding our words empty and unconvincing. Purpose is not expressed that way.

Purpose is only truly meaningful to the man on the scene who has at his command the means of accomplishing something and is caught up, engrossed, in getting a task done. Such a man is not objective; indeed, he is vehemently biased. But this is the kind of bias we so desperately need. He ignores most of the facts, and in so doing has time to attend to his job. He is too preoccupied to fill in all the data forms; but when he communicates, the message both demands attention and clarifies by its clear relevance.

Purpose *is* the man on the scene, struggling with a task, who can get direct feedback from his actions, who has at hand the measure of his accomplishment or failure. I do not think you can separate the two. Separate the man from the means for accomplishing his task, and purpose goes also. Remove him from the scene, from the source of direct feedback, and you remove purpose as well.

In designing information systems for command and control there seems to be no attention given to conveying purpose. How is purpose conveyed? Purpose is conveyed by saying what the alternatives are, not by evaluating their cost effectiveness. Purpose is conveyed by saying what the measures of effectiveness will be. Purpose is conveyed by asking questions, not for the sake of the answer, but to be sure the man questioned got the gist of your order. Purpose is conveyed by leadership and follow through.

The systemization of human knowledge has grown to such an extent that we find ourselves entrapped in a great web in which it seems that any strand we pull ties in with all the others tugging us in ways we did not anticipate. We tend to replace the question as to which way we *wish* to go by the question as to which way we *can* go. And we assume this latter question can be solved by simply getting our facts untangled, tracing through the scientific web of cause and effect, and thus clarifying the *old* alternatives and their cost effectiveness.

The realities with which the military must deal today are beyond such scientific explana-

tion. I am not talking about the "intangibles," nor about the role of "judgement," nor any of the other clichés used to justify intuitive behavior. The self-acceleration of context, in particular of our scientific/military context, has reached the point where changes are taking place in the system faster than our ability to properly take them into account. I have stressed above that fractionization is occurring, and the essence of fractionization is that overall rationalization of the total system becomes more and more impossible.

If science is inadequate for the direction of our military programs, by what should we supplement science? Under these conditions of fractionating context and decay of overall informativeness we must find some means to reestablish relevance and spur and inspire purposeful action with a unity of direction. Our current attempt to gather all possible information into one place and then use scientific methods to reduce it to a point where decisions can be made is resulting in just the opposite of what is required.

Indeed, the very cry of our operations analysts for clearer statement of requirements points up the almost total loss of measures of relevance and sense of purpose.

Let us recognize that some men have a better sense than others for where we are going and what can be done about it; and that some men have the capability to inspire and lead. Just as we look for the finest intellects to become scientists, we should similarly look for and esteem our men of judgement and our men of leadership. How much money are we spending today to encourage those who rate high on the college board examination compared to the incentive we are giving to courage of conviction, personal initiative, and the qualities of leadership? We do not breed leadership by centralization of decision making, by defiling the image of those who have achieved prominence as leaders and men of judgement by casually reversing their decisions, by giving precedence to the mathematically encapsulated intuition of men whose judgement has never been tested in operational situations, by occupying the time of our field officers with routine paperwork and the exclusion of honest tests of their effectiveness in successful operation and sustained operational preparedness.

It is in the practices of command and control that we determine the quantity and the quality of judgement and leadership that we shall be able to call upon in the future. It is the practices of command and control that allow the quantity and quality of judgement and leadership we now have to be exercised effectively. And it is the quantity and the quality of judgement and leadership that will determine whether the exploding context will fractionate in an orderly fashion, giving us the defensive capability that will sustain our great country, or will hasten the catastrophic fractionization that is now rotting our defensive capability from within.

To summarize:

- the requirements for communication must be minimized and the relevance of communication must be maximized by placing the prerogatives of command and the responsibility for reporting upward in the hands of the responsible official on the scene;
- the interconsistency of context must be maintained by keeping close to reality, by delegation to the point where feedback from the real world can be felt, by follow through in insuring that things are as they are reported to be, and by facing up to realities that we know but are now ignoring;
- relevance and purpose must be stimulated by attention and esteem for judgement and leadership and by the stimulation of these qualities by allowing them to be exercised at all levels of command.

These three principles are the central issues of command and control today. Our attention to them will mark the success or failure of the establishment of effective command and control across our military organizations. Do not be misled that the only problems that remain are how to automate decision making, store and retrieve documents or harden communication centers. Let us turn now to these central issues and what our industry can do to resolve them.

IMPLICATIONS FOR THE COMPUTER INDUSTRY

Exploitation of the Adaptability of the Computer

The development of the digital computer has provided us with a machine whose combination of adaptability and computational power is way beyond anything previously available. As a means of manipulating great quantities of data and carrying out extremely complex data processing, the computer opens whole new avenues by which we can bring to bear distant observations on current concerns. Furthermore, the high speed, stored program computer has the potential of placing these capabilities at our fingertips.

However, the exploitation of these two great virtues of the computer cannot be carried out independently. There is a trade-off between the computer as an adaptable instrument and the computer as a data processing instrument. To make the most efficient use of the computer as a data processing instrument requires efficient programming, minimizing of housekeeping and input-output functions, and careful scheduling of the computer's utilization. To make the most efficient use of the adaptability of the computer requires, in contrast, that the computer respond directly to the sometimes ill thought out instruction of the user, with much time spent on input and output to deal with the elaborate redundancies and conventions by which we normally communicate, and that the computer essentially wait for the user rather than the other way around.

Now the principal uses of the computer today are in applications where the maximization of its data processing capabilities are required. It is indicative to recall that the biggest sources of funds by far for the development of computers have been the Atomic Energy Commission and the data handling aspects of military intelligence. In both cases the emphasis is almost entirely on speed of processing. Even the output required is in the concise and sophisticated language of mathematics. No wonder the current generation of programmers and computer engineers place efficiency of computation and efficient use of the computer ahead of all other measures in deciding the appropriate way to apply computers.

Further, when the problem that the computer programmers and computer engineers see are so very large and complex and so distant from the underlying uncertainties giving rise to the computation, the idea that the data are suspect and the answer only an indication of what new question to ask, immediately requiring a new program, or second guess at inputs, is foreign, indeed. Thus the adaptability of the computer, which makes it such an ideal tool of conceptual exploration, has been downgraded. Little has been done until recently toward exploiting the other half of the computer's great capabilities. Indeed, relative to total expenditures or total population in our industry, little is being done today.

In particular, our command and control systems are not responsive to the rapidly changing environment, capabilities and objectives of the user. In fact, the principal result of all the push in command and control to date is the proliferation of organizations inserted between the ultimate user and his computer. There is the EDP office of the using command, his counterpart in the procuring command, the system monitors, the contractor's project office and finally the programming and design people. It is in the interest of each of these offices to seal the system requirements years before its operational date and see to it that the most efficient—and thus necessarily the most immutable—configuration of programs and equipment are provided. For example, soon after the initiation of one of the Air Force L systems, it became evident that the requirements stated in the request for proposal (by then 15 months old) did not take cognizance of the user's trends and plans for reorganization. The system contractor duly recommended a rather complete reorientation in his effort. Not only was this recommendation ignored, but 10 months later, 6 months after a reorganization of the user that made the work statement obsolete, the system contractor was required to deliver a detailed systems design and establish by detailed analysis that it filled, but did not go beyond, obsolete requirements of a nonexistent organization.

In the first part of this talk, I stress the growing problem of catastrophic fractionization of the context of the Department of Defense. I am convinced that a by no means insignificant contribution to this impending catastrophe is being made by past and current efforts in command and control systems. The huge, inflexible data systems with their insatiable demands on the field for irrelevant, often unused data, the cumbersome automation of decision processes whose development time cannot keep pace with changing requirements are contributing not only to the inflexibility of our defense posture, but immeasurably contributing to the inconsistencies they are supposedly resolving. There is nothing easier to corrupt than a complex, highly formalized data system. And there is nothing for which the coercions to corruption are stronger. I shall never forget the conversation I had some years ago with one of the leading contributors in our industry. At the time he was at a Naval shipyard, in charge of their computer installation. He described the problem there as having nothing to do with efficient methods of inventory control or production scheduling. Rather their then current data system had been so completely corrupted at the grass roots level that one could place no reliability on any available data, and as a consequence, the yard management was in a very real sense helpless in face of the working level control of shop foremen. How much farther has this gone today?

The real damage, though, is not being done by the command and control systems that are working; there are too few of these. It is the great promises and strange delusions that in a centralized data system we have the panacea of all our command and control problems. Mr. Gilpatric, when Assistant Secretary of Defense, said, "The machinery for gathering, analyzing, and presenting the data necessary for decision-making has, due largely to the extensive and imaginative use of automatic data processing by the military, advanced to a point where centralized decision-making is both efficient and effective." Mr. Esterly Page, Director, National Military Command System (Technical Support), repeating the above quote from Gilpatric, went on to elaborate on the choice that has been made: "During the past two years this nation has undergone a basic change in the philosophy of command and control which has resulted in new emphasis on the requirement for large quantities of finite and precise information to form the basis for consideration at the highest level of problems that previously were considered and unfortunately decided at a much lower one." Like the wooden walls of Athens, computers may be the key to our survival. But what happens when those high up in the lonely citadel find they chose the wrong wooden walls, and the ones they chose have been rotted away by the proper corruption which seeps in when the traditional coercions for responsibility have been taken away?

However, there is a bright spot in this otherwise disturbing picture. The Advanced Research Projects Agency is supporting a number of well-chosen programs for the development and exploitation of the adaptive capabilities of computers. Certainly one of the most ambitious of these programs is Project MAC, undertaken by the Massachusetts Institute of Technology. The Air Force is also turning toward developments along somewhat similar lines, as was stated by General Terhune[3] at the Fall Joint Computer Conference in Las Vegas.* There are, as well, programs sponsored by other agencies, both public and private. For example, the commercially supported project undertaken by the Control Data Corporation concerning computer supported engineering design.

These projects have as their general objective making the computer directly available to the individual researcher, analyst, and staff officer, as an adjunct to his creative thinking. The technical objectives are threefold. First, by means of remote consoles, interrupt features, and associated software, to provide a number of users direct and parallel access to the computer. Second, by means of problem oriented languages, including graphic and natural languages, to provide access to and response from the computer in a form that is natural and effi-

* This point of view was anticipated by "A Concept for the Navy Operational Control Complex of the Future," by R. W. Callan and F. B. Thompson,[1] and "Fundamentals Underlying Military Information Systems Design," by F. B. Thompson, a paper presented at the First Congress on the Information Systems Sciences, November 20, 1962.[2]

cient for the user. Third, by organization and advanced concepts of software logic, to provide more and more intelligent responses to an increasing variety of possible requests. These capabilities will take the computer from down the hall and behind the counter, and place it as a direct adjunct of the staff officer.

Command and Control Implications of Direct Access Systems

The general nature of these technological developments is probably known to you. What I should like to bring into sharp focus here is the implications these technological developments hold for the central issues of command and control. A computer as a direct adjunct of the staff officer is a very different thing than a computer as the central instrument of a great data system. Those who speak of direct access systems often describe how they will allow the staff officer to make use of some central data base, and the implication seems to be that the staff officer will have direct access to the great assemblage of fact that now stands assembled but inaccessible by the great data system. This is a false view. Access to a base of fact can be meaningfully accomplished only if the user has a feel for the relevance, the source, and the units of those facts. In a very real sense, they must be his facts, his data. We too often assume that the number of missiles in Cuba, or the number of atomic weapons we could deliver to enemy targets are sufficiently well defined when in fact their uncritical application by the uninitiated is fraught with danger.

This point needs to be stressed. Meaningful data is a fragile, subjective thing. The intelligent analyst knows this when he insists on knowing the use that is going to be made of his estimate before he is able to produce it. The contracting officer knows this when he refuses to allow even a unilateral passage of information during deliberations on a source selection. The combat commander knows this when he gets out among the troops to sense the situation on the spot. One can reasonably and responsibly use data only if it is one's own data.

There is a second aspect of this matter of the data base underlying these direct access systems. The amount of data that *could* be accumulated in, say, a major command headquarters is astronomical. The choice of just what data *is* to be found in the system thus becomes an important command decision in itself. That choice embodies the sense of relevance of the headquarters. Further, the implementation of that choice by directives on subordinates and requests to lateral and higher headquarters, is a principal means of conveying purpose and establishing direction. The headquarters whose reporting requirements are static and all-inclusive gives up one of its principal tools of control and opens the door to subversion of its leadership. It is the direct access system that can give new vitality to data as a sensitive instrument of command and control.

Therefore, the implication of the technological development of direct access systems is that there will be closely knit staffs addressing relatively small but dynamic bases of highly relevant data as they deal intensively and purposely with problems local to their scene. Such direct access systems cannot be reasonably justified on any other grounds.

Thus the technological development of the computer as an adaptable instrument, responding directly to the requirements and local context of the user marks the beginning of the counter trend. Not only are the continuing failures of the automated information systems becoming more and more obvious, but the seeds of a new type of system suited to the needs of the purposeful headquarters on the scene are being sown. The catastrophic fractionization of the Defense Department is being accelerated by the obsolete centralized information systems. It is the latter, the system that can be used responsively by the purposeful staff, that can bring the necessary vitality and responsiveness to orderly fractionization.

The Promise and the Needs of the Highly Responsive Staff

Throughout this paper, the underlying problem has been how to deal with the exploding military context. We have emphasized (1) the need to place the prerogatives of command and responsibility for reporting upward in the hands of the responsible officer on the scene; (2) the need to maintain a direct hold of

reality; (3) the need to establish purpose and measures of relevance through leadership. How can this be done, say in the environment of military command?

The essence of orderly fractionization is that the units of command must be in a position to maintain a viable context, a cohesive and consistent view that integrates their capabilities, environment and objectives in a way that is responsive to the realities with which they must deal. The commander and his staff must know what he wants to do, what it will take to do it and what he has to do it with, and they must have a sense of readiness that his objectives, his requirements and his capabilities add up with possibly a margin of capability in reserve. No one of these three ingredients of his context—objectives, situation and capabilities —is given to him immutable. He must be able to modify his detailed objectives, reassess his situation, reorganize and redeploy his capabilities. That is, he must maintain the consistency of his context of command. But more than that, he must make sure of this intelligence, test and exercise his forces, prod and react to his superiors; he must maintain the validity of his context of command. This maintenance of context is the essence of staff work and the foundation of command and control. The orderly fractionization of command is tantamount to the establishment and reestablishment of a hierarchy of command in which each headquarters *can* maintain such a context peculiar and appropriate to its position in the hierarchy.

As the rate of expansion of overall context increases, this hierarchy of command must expand and deepen. Each element in turn must increase the efficiency and maintain the relevancy of its context. If this is done there will indeed be fractionization, for the contexts of forward elements, in touch with the rapidly shifting local situations, will responsively shift and move in ways the contexts of higher elements cannot moment by moment respond. These higher elements, however, operating at higher levels of abstraction, will follow at their appropriate rate, their appropriate realities, of their appropriate scene. Fractionization of the overall context inevitably will occur.

Let me restate that in the bluntest possible terms: In the Department of Defense today, inconsistencies between the contexts of the various elements of the organization are inevitable. It is the character and source of these inconsistencies that must be controlled. If that fractionization is realized in terms of the responsiveness of the disparate elements of the command hierarchy, it will accommodate greater and greater responsiveness to change, and permit an orderly control of a vast and vastly informed organization. Consistency of overall context cannot be maintained in the face of change. But orderly fractionization of context that places the prerogatives of control at the point where the sense of change is most meaningful, that insures the degrees of change will be felt at the appropriate level and that inserts purpose into change, such orderly fractionization can permit our organizations to live and grow and prosper.

In Summary

Thus the crucial informational problem is the maintenance of relevant, consistent, valid context in each separate, disparate organization or military headquarters. This is the central task of staff work, the essence of command and control. It is in the perspective of this task that our information systems must be planned and developed. The appropriate information system must be a direct tool of staff, as natural and direct to use as the telephone. The data held by the system must be put there by the staff, must be the intimate changing product of the staff, the factual base of *their* context whose relevance is *their* most crucial task. And that system must be the staff's system, not a universal system, not a standard system, but one that is recognized as what it is, inconsistent with all others, to a degree irrelevant to all others, and thus free to be responsive to the sense of relevance of its staff, the purpose of its commander or administrator. From such a context, communication will be relevant. And this communication, by its very shifts and changing emphasis, will convey purpose and direction as well as fact.

This is a very different kind of system than the great data systems and automated control systems. It is a system that allows direct access in natural language, it is a system whose files

and formats, whose inputs and vocabulary are always changing, continuously changing.

The Department of Defense is tending toward catastrophic fractionization. The reversal of the process of this explosion may still be possible. It rests in the hands of two groups: First, those in control of defense policy, in particular those who are determining the policies for military command and control. Second, it rests on us, for it is our industry that can turn to the development of systems that will exploit the adaptability of computers and will place them as the tools of staff in maintaining relevant, responsive contexts of command. The challenge to realize orderly fractionization is very real and urgent today, a challenge to which we must respond.

REFERENCES

1. CALLAN, R. W., and F. B. THOMPSON, "A Concept for the Navy Operational Control Complex of the Future" (General Electric, TEMPO, RM62 TMP-49, July 1962).
2. THOMPSON, F. B., "Fundamentals Underlying Military Information Systems Design" (General Electric, TEMPO, SP-183, November 1962), to be published in *Military Information Systems*, edited by Edward Bennett (Praeger, New York, in press).
3. TERHUNE, T. H. An Address to the American Federation of Information Processing Societies, Fall Joint Computer Conference, Las Vegas, Nevada, November 12, 1963.

SOME OBSERVATIONS CONCERNING LARGE PROGRAMMING EFFORTS

Almon E. Daniels
Weapons Systems Evaluation Division
Institute for Defense Analyses

INTRODUCTION

The requirement to construct large systems involving both men and computers to assist in the evaluational and decision making processes of command and control leads very quickly to the development of large and complex computer programs. While my own efforts have been largely devoted to the development of detailed computer simulations of strategic air warfare,[1] I have had an opportunity to follow in more or less detail the progress on a few of the command and control systems. The observations which are made here relate to a broad class of computer data systems and certainly apply to the ones evolving in the command and control area.

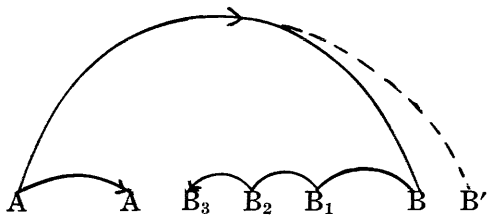
One may think of a computer program as a progression of smaller programs or routines. Among these routines are ones which alter data presented to them, perform calculations, make new arrangements or combinations of the data, provide for employing alternate paths in the program, etc. Of considerable importance also are the control routines which bring together at the proper time the correct routine with the data on which it is to operate. In more complex programs, the sequence of use of the routines and data is subject to modification by external events.

The tougher problems in the design of computer programs begin to arise when the internal memory of the computer is too small to contain all of the routines and data involved. For the large man-machine systems the internal memory might be likened in size to a teacup used to dip a mixture of program and data from a barrel. Much larger memories will simplify some of the problems, but they will not eliminate nearly all of them.

These remarks have been made to clarify, for the present purposes, the meaning of the expression "large computer program."

The development of a large system goes through several phases. First, there is a formulation of system requirements. Then follows a period of system analysis, problem definition, and design. The computer hardware is selected, a detailed design is made and implementation is undertaken with numerous changes and delays. Experimental applications are made, redesign occurs, and modifications of the computer program are made. Further trials are made, and the cycle is repeated.

The elapsed time for the accomplishment of these phases is measured in years. The design and implementation of most of the Air Force "Big L" systems have extended for three to five years. The other services have faced similar schedules. Under very favorable circumstances, a substantial application can be brought to a productive state in about two years.



DESIGNERS' DILEMMA

Designers of large data systems begin at condition A and envision a solution B, hurdling a number of intermediate steps. Because of the anticipated elapsed time of three to five years, embarking on the task of attaining solution B without intermediate steps requires the acceptance of a fairly bold assumption. It has to be assumed that solution B, which has been arrived at in terms of present day problems, will in fact be suitable for the intended task when the system is operational. In other words, the assumption must be made that the problem and the solution will remain relatively constant. Experience has shown that this happy state does not exist. The more distant the goal, the less certain the solution becomes. Furthermore, because the problem exists at condition A, there is pressure to arrive at a useable solution as soon as practicable. Consequently, as work progresses, several compromises are made. Call these less elegant solutions in succession B_1, B_2, and B_3. As time passes, additional knowledge is acquired, and the actual requirements of the problem change to B′ while the compromise solution is still B_3. A person not troubled by the esthetics of pure design could set out toward the not-crystal-clear goal B by making the step from A to A′ (which may be rather close to B_3) with much greater chance of early accomplishment of the lesser goal.

Our experience with computer applications verifies the comment of one keen observer in the profession: "When we cease to change a program, we cease to use it."

If there is any conclusion to be drawn from considering these two ideas together, it is the following: Make as complete a system study as possible, but design the implementation in such a way that a portion of the system can be exercised usefully at an early date. One must attempt to build the system and its modifications so that the user can operate the system while its usefulness is being extended incrementally. It is almost inevitable that exhibiting sample products of the system will lead to changing the design of the system; and the sooner this restatement of the requirement occurs, the sooner the first round of modifications to the system can be undertaken. Changes will be required on a continuing basis as the users of the system better understand what the system can do for them.

USER PARTICIPATION

It is next to impossible for the system analysis team to discover the full extent of the system being studied or to find all the boundary conditions unless the cooperating representatives of the user are sufficiently experienced in both the operational and computer aspects of the application to exhibit all the pertinent facets of the problem. The distinction is here made between the ultimate user of the system and his representatives who are assigned to assist with the design and incremental implementation of the system. The backgrounds and perspectives of the representatives may result in the incorporation of some undesirable features in the system, if they are not able to project their thinking to the needs of the ultimate user. If the ultimate user can be involved in the early exercising of the incomplete system, clarification as to emphasis and detail should occur before the whole design is implemented in an unsatisfactory fashion.

Despite the possibility that guidance from representatives of the ultimate user may introduce some ideas into the design which will have to be modified at a later date, the participation of carefully selected representatives of the user throughout the design, implementation, and testing of the data system is vital.

Without such participation, the user's organization will be ill-prepared to understand the series of compromises which have been made to bring the system to an operational state, and will not understand that asking for too much in a hurry is likely to delay the completion of

the initial portion of the system and its incremental extensions.

The user's personnel must be able to undertake the maintenance and modification of the system after the team from outside the organization has completed its work. In order to supply this capability, the user organization must provide sufficient personnel of suitable talent to participate in the implementation of the system. These participants must be deeply involved in translating the design to language acceptable to the computer, debugging the routines created, developing test problems of proper sophistication, producing the documentation, etc. As the key personnel assigned this duty may require two years or more to become proficient in substantial portions of the system, care must be taken to assure continuity in their assignments. Until the problem of rapid turnover among computer programmers becomes less acute, it is likely that most of the more stable personnel involved will be among those in the employ of the using organization.

TESTING

For computer programs of considerable size, it is useful to distinguish three levels of testing. The programmer devises a test or tests for each subroutine he builds in order to assure that the computer reacts as anticipated to all the circumstances he considered in designing the subroutine.

As these building blocks are completed, they are ready for testing in combination with other subroutines and other aspects of their intended environment. Their input and output conventions are checked for compatibility. Larger test procedures are devised to exercise the joint functions of the program complex. Appropriate adjustments are made in the program until all anticipated results are produced.

The third stage of program testing[2] consists of putting the program into limited operation to observe its behavior when dealing with the many variations of data which arise in practice. It is very difficult to devise test problems which employ all the possible paths in a program. Even though the program responds correctly in a limited test, it is possible after several weeks of successful operation for the program to encounter some new combination of inputs against which it was not adequately protected.

DOCUMENTATION

Documentation of the system created in one of these three-to-five-year projects is an important though time-consuming endeavor. The description of the program in the language accepted by the computer is a key part of this record and must be kept up to date as program changes are made. This level of documentation is the "court of last resort" in isolating precisely what some routine does, but the technical remarks or diary prepared by the programmer while he has all the facts in mind will reduce the amount of detective work which must be done some months later when an unanticipated change must be made. Two or three other descriptions of the system may also be required at decreasing levels of detail.

Adequate documentation is important to the person who created the computer program, as even he soon finds the maze of detail confusing without rather complete records. The person who creates one program is never free of it unless he makes it possible for others to modify his program without performing major research. The documentation must be generated so that a given programmer may be assigned to a different project. This documentation is essential, for the protection of the user, as the programmer may at some critical time no longer be available for consultation.

SIMULATION

Simple exercising of a large man-machine data system with the use of a single bank of information in a fixed pattern will bring into focus the utility of such a system to aid military decision makers only under rather limited circumstances. A simulation technique should be superimposed to alter the data flow according to various assumed alliances, postures, strategies, or the like. Such simulation exercises of the system in the operational environment by users of appropriate echelons will give

them synthetic experience about situations which have not been encountered in sufficient detail in any other medium. Only by exercising the system with data flowing at an accelerated pace will many of its weaknesses be discovered. The simulation must, of course, incorporate uncertainty, conflicting information, degraded communications, etc., if realism is to be obtained.

Considerable effort has already been devoted to the development of detailed computer simulations of the air interactions in a nuclear exchange.[3] It would be hoped that these simulations might be used to generate under various assumptions much of the information required to alter the behavior of the man-machine data system.

It should not be assumed that providing the simulated environment to influence the large data system is a small undertaking. Depending upon the detail introduced, the project may be even larger than the original one.

Thus far this discussion has dealt with the notions of system design, incremental implementation, inevitability of modifications, user participation throughout the project, testing at three levels, integrating documentation with the development of the program, and the usefulness of providing a simulation environment for the data system.

It also seems appropriate to record some additional remarks based on participation in or observation of several large programming efforts. These remarks will range over the topics: level of effort, estimation of time required, hardware considerations, reaction to frustration, and programming systems. While these topics are not clearly related, the comments are intended to provide at least a partial check list for those who are brave enough to undertake the development of a large computer program.

LEVEL OF EFFORT

It requires great skill and unusual circumstances for a large system of programs to arrive at a satisfactory operational state in less than three years. Even though the period since 1955 has provided several instances of this bit of truth, every new group which embarks upon a new project is dominated by persons who believe that they can avoid the mistakes made on other projects and shorten their own schedule. To some extent they do profit from the experience of others, but they usually make a few new mistakes of their own.

It is important to understand that a small team of four or five experienced analysts can accomplish about as much in the first four months as a team of twenty. Toward the end of the initial phase, the small team should be supplemented by specialists in various areas as required. There will be no advantage in gathering a team of 100 technicians at the beginning of the effort. As a matter of fact, such an action will probably delay the project.

A healthy growth in the effort applied to the project is important. A modest beginning with thorough indoctrination in the early months of the user-contractor relationship will pay off in the end.

Wide fluctuations in the support given the contractor will cause serious difficulties. If enough money is provided to acquire too large an increment of personnel, most of the old hands will be devoting too much time to training the new arrivals. If too little support is provided, progress lags and morale suffers. As new arrivals on the project will not contribute much for several months, the project is relatively insensitive, in the short run, to small personnel increases.

ESTIMATION OF TIME REQUIRED

Aside from the scarcity of experienced personnel capable of performing the scientific and engineering tasks involved in the development of a large computer data system, there are some administrative problems which lead to fixing the minimum time at two years regardless of the complexity of the system. Among the administrative problems are the following: budget justification and funding release practices, site construction schedules, and equipment delivery schedules.

It has been customary in some quarters to estimate the productivity of skilled programmers at 3000 instructions per man year. When

a great deal of on-the-job training is involved, the estimate should be set at about 1000 instructions per man year. When smaller teams are given the responsibility for well defined sub-projects, instances of 12,000 to 15,000 instructions per man year have been observed.

In estimating costs, it appears necessary to allow for each programmer two other persons classed as analysts, administrators, or engineering support personnel.

The more complex data systems fall in the 100,000 to 200,000 instruction class.[4] Using the factors just given, one can begin to estimate the manpower required and the cost of implementing such systems. By imposing limits on the size of the team to be involved and taking into consideration the funding schedule, it is then possible to obtain a first approximation for a completion date.

After the design is relatively complete, the users and designers must agree that the design meets the requirements of the project. After the detailed programming effort begins, a major re-orientation of emphasis which requires a redesign of a major segment of the system will normally delay considerably the completion of the project.

Once agreement has been reached on the design of a specific large computer program, the user must exercise a great deal of patience and forbearance. No amount of exhortation will materially affect the progress of the project and may even cause delays. The user who schedules other events very closely on the basis of the promised completion date of a large programming effort is almost certain to have major disappointments.

Some attempts have been made to apply PERT techniques to programming efforts. Until estimating techniques in the programming field are more reliable and the addition of extra manpower on lagging subroutines has a more predictable effect in the short run, the development of a PERT network will serve only to help the managerial echelons to understand the relationship of the various pieces of the program being constructed and to make the initial assignment of effort more intelligently.

HARDWARE CONSIDERATIONS

Rapid strides are still being made in computer hardware development, and significant improvements as to speed, reliability, capacity, and flexibility are available every two or three years. It has been pointed out that the design and programming effort for large problems takes a minimum of two years.

It is hard to imagine the confusion which would arise if a computer, of not too well defined characteristics, were being constructed for a large problem which was being programmed at the same time. First, it would require most of the hardware construction period to create the sophisticated programming system required to permit the large problem to be accepted by the computer. Further, about half way through the period, there would be a requirement to debug some of the routines of the large problem.

As the problem has probably changed materially during the programming period and most certainly will change as soon as the program has been used a few times, most persons who have considered these facts have agreed that nothing is to be gained by trying to specify the characteristics of a large-scale *special purpose* computer. If it is built to satisfy the requirements of a problem as defined now, there is a good chance that it will not be adequate for the problem as it is understood shortly after the computer is completed three years from now.

A useful competition among the computer hardware manufacturers is in progress. For reasons which need not be considered here, different types of computers are being selected for installation at various locations on th basis of current requirements with well-documented justifications. Much of what is to be done at location A on computer X may also at some time have to be done at location B on computer Y, etc. If a programming system implemented on computers X and Y would, in fact, accept the programs first written for the other computer with only modest changes, considerable savings would result.

REACTION TO FRUSTRATION

Any person who has the tenacity to spend several weeks trying independently different approaches to the solution of a puzzle before success can appreciate in some small measure the frustrations of the programming activity and the final feeling of personal triumph when the last important programming bug is corrected. As the process of preparing a large program often extends over more than a year and the mistakes of many people are involved, the frustrations are of a higher order. In the normal course of events, nerves become frayed, fingers are pointed, neuroses appear, and other psychological and physiological reactions are not unknown. A few persons on the project who accept the ultimate technical responsibility for the completion of the project will find the pressures unabated for a period of months. It is small wonder that persons, who have experienced this trauma a time or two, have fairly firm ideas as to the circumstances under which they will accept such a responsibility again. The completion of the project will lag unless this concentrated effort does occur, and the managerial chain on the project cannot, at the beginning, predict which members of the team will punish themselves in this fashion.

One of the important contributions being made by the more powerful programming systems is the elimination of many of the frequent recording and cross-referencing errors so that the more important logical blunders do not remain concealed for long in the debugging process. This alone materially reduces the length of the effort and permits more concentration on the solution of the important problems.

PROGRAMMING SYSTEMS

Because of the general confusion in connection with the use of the words "programming system," it may be helpful to distinguish (1) programming languages, (2) compilers, and (3) operating systems.[5, 6]

A programming language is the set of symbols and conventions which are designed for the convenience of the programmer to express his detailed solution to the logical, manipulative, and computational problems encountered.

A compiler translates the expressions of the programming language into sequences of instructions in the basic language of the computer hardware.

An operating system augments the capabilities of the computer in such a fashion as to relieve the programmer of the requirement to provide in detail for input/output scheduling and assignment, memory allocation, etc., so that the programmer need not resolve these recurring problems in numerous variations and can devote his efforts to implementing the data system design.

The JOVIAL system includes all three of these components and has been implemented on several large-scale computers, but the versions of the language are not identical.[7]

NELIAC has been widely used and consists of a language and a compiler. The data routines are incorporated during the compilation. This technique has permitted the larger computers to compile programs for computers with memories too small to accommodate a compiler.[8]

The CL-II programming system consists only of a compiler and an operating system. The compiler is described as "syntax-directed" and can accept any of the present algebraic languages for which it is provided the necessary input tables. It has been demonstrated that careful preparation of these tables enables the compiler to generate as compact code as an experienced programmer. The sophisticated operating system of CL-II is based on the concept of the "extended machine" first expressed by Holt and Turanski.[9, 10]

A good programming system should

(a) Encourage modular construction of the computer programs.

(b) Provide for data descriptions which are independent of the program until compilation occurs.

(c) Relieve the programmers of the necessity of constructing substantial debugging environments.

(d) Eliminate the need for developing elaborate control programs.

There seems to be no way to avoid all the human errors which are made in the course of developing a large computer program, but a good programming system can relieve the programmer of many of the housekeeping problems so that his thoughts can be directed to the peculiarities of the data system being automated.

If there is to be reasonable progress in the production of a large computer data system, the programmer's work should be reduced as much as possible, and the initial system should be constructed with the anticipation that changes and extensions of the system will be required on a continuing basis.

SUMMARY

In closing, it seems worthwhile to summarize the main points of this discussion:

(a) The users of the data system must collaborate with the technicians throughout the design, construction, and testing of the system.

(b) A full design is desirable, but a path should be chosen through this design so that a useful sub-system is available as soon as possible.

(c) Anticipate changes and extensions to the system.

(d) Do not stint on the documentation.

(e) Exercising the system fully will probably require building a simulated environment.

(f) Putting extra manpower on a project at the wrong time to accelerate the effort will usually accomplish little except to heighten the disappointment.

(g) Large-scale special purpose computers still seem inadvisable because of changing requirements.

(h) The programmer who takes his profession seriously does not find it to be an easy life.

(i) Programming systems may not have reached perfection, but they offer the programmer some relief from many of the troublesome housekeeping details.

(j) Standardizing on a single algebraic programming language is apparently no longer required.

No claim is made that these observations form an exhaustive list of all matters to be considered by a group embarking upon the development of a large computer data system to assist in the evaluational and decision making processes of command and control. Each person who has been involved in such an effort will have a few items to add. Perhaps even this partial list will help some group to avoid a few of the mistakes others have made.

REFERENCES

1. DANIELS, A. E., "The User's Viewpoint on the Creation and Application of a Large Simulation Model," Proceedings, First War Gaming Symposium, Washington Operations Research Council, 30 November 1961.

2. SAMS, BURNETT H., "Some Observations on the Development of Large Programs," First Congress on the Information System Sciences, Air Force Electronic Systems Division and The MITRE Corporation, November 1962.

3. PENNINGTON, A. W., "A Description of the STAGE System," Proceedings, 11th Military Operations Research Symposia, April 1963.

4. HEINZE, K., N. CLAUSSEN, and V. LABOLLE, "Management of Computer Programming for Command and Control Systems," System Development Corporation, 8 May 1963.

5. KROGER, MARLIN G., et al., "Computers in Command and Control," Institute for Defense Analyses, Resarch and Engineering Support Division, Technical Report Number 61–12, November 1961.

6. HAVERTY, J. P., and R. L. PATRICK, "Programming Languages and Standardization in Command and Control," RM–3447–PR, The RAND Corporation, January 1963.

7. SHAW, C. J., "The JOVIAL Manual, Part 3: The JOVIAL Primer," TM–555/003/00,

System Development Corporation, 26 December 1961.

8. HALSTEAD, M. H., "Machine Independent Computer Programming," Spartan Books, 1962.
9. HOLT, A. W., and W. J. TURANSKI, "Man to Machine Communication and Automatic Code Translation," Proceedings of the Western Joint Computer Conference, 1960.
10. CHEATHAM, T. E., JR., and GENE F. LEONARD, "An Introduction to the CL–II Programming System," CA–63–7–SD, Computer Associates, Inc., Wakefield, Mass., November 1963.

SOME COST CONTRIBUTORS TO LARGE-SCALE PROGRAMS

Burt Nanus and Leonard Farr
System Development Corporation
Santa Monica, California

INTRODUCTION

In the early days of computer technology, only a small handful of highly competent, scientifically-oriented researchers were familiar with the programming arts. In those days, the management of programming effort was only slightly different from the management of other types of research activities. Each project was unique and its probability of success uncertain; experience was severely limited; tools and techniques were custom-built for each job. In the computer field today, despite many thousands of man years of experience in program development, we still tend to plan and to manage as if each program were a unique research project. This is partly due to the immaturity of the field, and partly because we have not yet fully recognized that the similarities between computer programs and their development are far more extensive than the differences between them. We have seen the development of many new tools to make the programmer more effective in his work; we must now ask whether we can develop new tools to make the program manager more effective in planning and organizing his scarce resources of talented manpower and expensive computer time.

One of the most important requirements for management planning is an accurate estimate of the resources required for the completion of the project. In programming management, the two principal resources to be estimated, scheduled and controlled are man months and computer hours. Together, these resources may be considered the cost of producing the program. Historically, these costs have been very poorly estimated; there are abundant examples of actual costs that exceeded estimated costs by 100 per cent or more.

Because better cost estimation is an important step toward more effective programming management, because the costs of programs may be a significant portion of the total costs of large management or command information systems and because the estimates have been little better than guesswork to date, the Advanced Research Projects Agency of the Office of the Director of Defense Research & Engineering sponsored some research in this area at the System Development Corporation (SDC). Early efforts were aimed at data collection and analysis of several large-scale command and control programming efforts representing a total of more than two million instructions and 1500 man years of work.[2, 3, 6, 7] This paper is a summary of a subsequent effort to (a) identify the common factors that influenced the cost of developing programs and (b) perform a preliminary analysis on some of the data.* These are viewed as necessary first steps toward the development of a more accurate cost esti-

* The full reseach report upon which this paper is based will be published as TM-1447, *Factors that Affect the Cost of Computer Programming*, System Development Corporation.

mating procedure. Further progress depends upon more systematic data collection, and one of the purposes of this paper is to recommend the types of data that should be collected by programming managers for estimation purposes.

PROBLEMS ENCOUNTERED IN DETERMINING COST FACTORS

In the process of identifying and analyzing cost factors, one soon encounters a number of limitations in the programming field that may not exist in other, more mature disciplines. Some of these problems include the following:

1. *Lack of Agreement on Terminology*—There are no universally acknowledged definitions of many of the terms used in the computer programming process. For example, the words "debugging," "parameter test" and "program validation" may all describe the same process; a "programmer" in one organization may be called a "coder" in another and a "system analyst" in a third. Although we attempted to keep within the more limited context of command and control programs in our research, we found, even in this narrower field of programming, a widespread lack of agreement on terminology.
2. *Poor Definition of Product Quality*—Apparently there has been little success in defining those attributes that characterize the nature or the quality of a computer program. For example, one hears programmers talking in terms of flexibility, economy of memory, and maintainability, but there seems to be no generally agreed upon criteria for comparing programs on the basis of these attributes.
3. *Poor Quality of Cost Data*—Present cost collection methods seem to be designed primarily for accounting purposes and not for planning or control. For example, the collections of costs are usually grouped by organizational units rather than by product or function to be performed.
4. *Dynamic Nature of the Field*—Although computer programming is maturing as a discipline, there is still a wide diversity of techniques and approaches being developed and used. As a result, any study of cost factors must consider the history and likely future trends of programming technology.
5. *Nonquantitative Nature of Some Factors*—Experience has shown that many of the factors that affect the cost of computer programs are qualitative in nature. In some cases, it is possible to predict at least the direction that cost will be affected by an increase in a given factor. For example, one would expect that the more experience one had with the particular type of program or computer involved in a given task, the less it would cost to perform that task. In other cases, qualitative factors appear to have a nonmonotonic effect as when an increase in a given factor (e.g., management planning) first decreases and then increases total cost. Of course, determination of the magnitude of the effect on cost of qualitative factors is even more difficult than determining the direction of the effect.

Although these problems combine to make an analysis of computer programming cost factors somewhat difficult, a start must be made if program development efforts are to be more effectively planned and managed.

COST FACTORS

It is possible to identify hundreds of factors that contribute to the cost of computer programs, if such a level of detail is desired. In this paper, we will present a list of approximately 50 such factors, consolidated from a much larger list. Obviously, some classification scheme is necessary for discussion or analysis purposes. Factors might be grouped by work phase, such as program design or test; by management activity, such as planning or evaluation; by general categories such as resources, requirements, or environment; by units of cost measurement, such as man months or dollars; or by the classic accounting method of direct and indirect costs. However, these schemes seemed to cause difficulties because of ambiguities and overlap; as a result, a new classifi-

cation scheme was developed and is illustrated in Table I, in which factors are divided into these categories: The Job to be Done, The Resources that are Available, and The Nature of the Working Environment.

Since a full discussion of all of the factors is not possible here, only the most important one or two in each category will be briefly presented; the complete list is included as an appendix.

TABLE I—COST FACTOR CLASSIFICATION SCHEME

Logical Grouping	Category Name	Category Definition
THE JOB TO BE DONE	1. Operational Requirements and Design	Includes cost factors associated with the operating characteristics of the system for which the program is being written.
	2. Program Design and Production	Includes cost factors associated with both support and operational programs as determined by the constraints imposed by personnel, hardware and operational requirements.
THE RESOURCES THAT ARE AVAILABLE	3. Data Processing Equipment	Includes cost factors associated with the data processing equipment required to produce and test a program, including all input, output and peripheral equipment.
	4. Programming Personnel	Includes cost factors resulting from the direct labor needed to completely develop a program.
THE NATURE OF THE WORKING ENVIRONMENT	5. Management Procedures	Includes cost factors associated with the plans, policies, practices and review techniques used in the administration of all phases of program development.
	6. Development Environment	Includes cost factors resulting from relationships of the programming staff with other organizations, such as customers and other contractors.
	7. Facilities, Services and Supplies	Includes cost factors related to supplies, physical plant, indirect labor and overhead.

THE JOB TO BE DONE

1. *Operational Requirements and Design*

The factors in this category tend to center around the question, "How well are the operational requirements of the information system known?" or, "How well is the problem defined?" Unfortunately, it is virtually impossible at the current state-of-the-art to evaluate the cost-contributing effect of many of the factors which relate to this question.

The primary cost factor in this category would appear to be the extent of innovation in the system, in its components, and especially in the automatic data-processing function. The extent of similarity of the new system to older systems may be a clue to estimating how clearly or easily the requirements of the new system can be stated. If the requirements are well known to the program designers, the programming job is more straightforward and less costly. Somewhat related to this factor is the extent to which the programming organization will participate in the formulation of requirements; the less active its role in determining requirements, the more likely misunderstandings will develop, resulting in costly errors, omissions and ambiguities.

2. *Program Design and Production*

In a large-scale program system, program design involves the determination of the broad logical subdivisions of the computer system, the design of an executive program to control the sequencing of programs, the design of the data base structure, the allocation of computer storage, and specifications for utility and support programs. As in the first category, these factors center around the question, "How clearly understood are the program requirements?" The factors also are concerned with the size and complexity of the job; the resources and tools available; and the plans for documenting, verifying and testing the product.

Undoubtedly, the most important cost factor in this category is the number of computer program instructions and the types of programs that must be produced. In current techniques for estimating cost, the size of the program is often used as an intermediate measure to estimate the number of men who will be assigned and the number of computer hours required. Despite this reliance on size as the key to cost estimation, it appears that little research has been done to develop systematic and reliable ways to predict the number of instructions.*[1] So far, the experience of many program managers is that the number of instructions is often grossly underestimated, except when very similar programs can be used for comparison.**[4, 8]

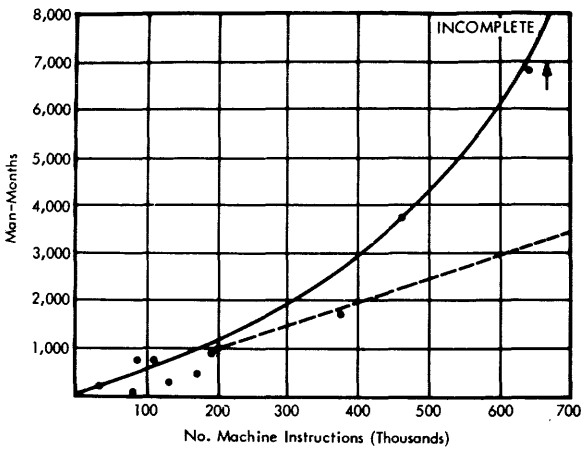


Figure 1. Man Months Versus Program Size for Eleven Large-Scale Programs.

The conversion of the estimate of number of instructions to programming man months for large-scale systems is frequently done by allowing one man month for each 200 instructions. To test this rule of thumb, empirical data on the number of machine language instructions in eleven large command and control systems were compared with the 200- instruction-per-man-month guideline (Figure 1). The programs represent a variety of command and control systems using several different computers and languages. The number of man months includes program design, testing and coding. A further analysis of the data revealed that the production rate for operational programs averaged 225 instructions per man month while the rate for utility programs was 311 instructions per man month. The explanation for the higher rate of utility program production was that the

* R. Bleier of SDC has reported on our attempt to relate total program length to the frequency of certain decision-class instructions, TM-1603.

** For example, a report of the Controller's Institute stated, ". . . almost all EDP groups have at one time or another seriously underestimated the number of steps required. . . . Every company we visited added a substantial safety factor varying from 20% for a company which claimed, due to experience, a reasonable accuracy in its estimating procedures, to 400% for a company which had found itself that far out on a previous estimate." (*Business Experience with Electronic Computers*, New York Controller's Institute Research Foundation). "Measuring the Profitability of a Computer System." See also J. D. W. James.

program developer is his own customer for the program system and, therefore, can write his own requirements with little external coordination. For smaller programs not shown on the figure (i.e., less than 10,000 instructions), rates of as much as 400 to 1000 instructions per man month for individual programs were reported.

The conversion of the estimate of number of instructions to computer hours is also subject to various rules of thumb. Figure 2 is based on experience with eight large programs. The three points falling below the line represent efforts in which a procedure-oriented language was used, indicating that such use may reduce the amount of machine time required for program development. As shown in the figure, one

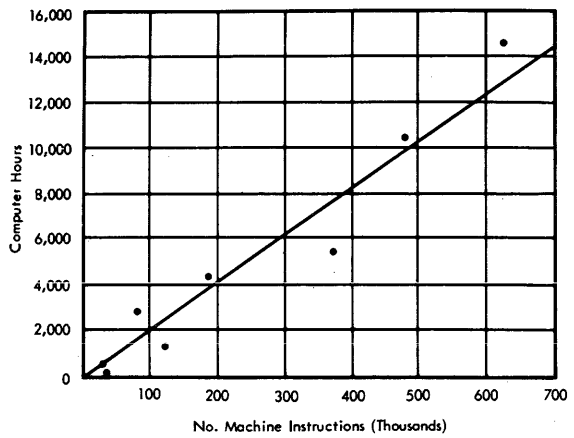


Figure 2. Computer Hours Used as a Function of Program Size for Eight Large-Scale Programs.

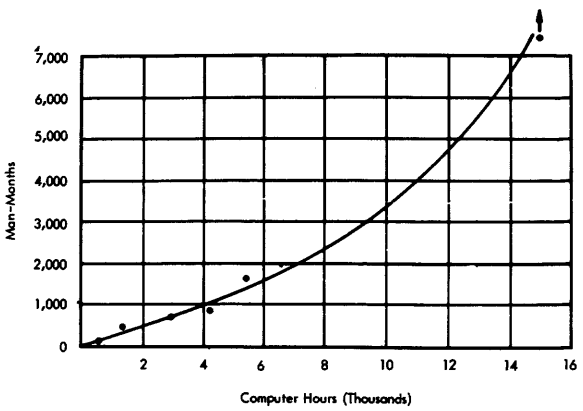


Figure 3. Man Months Versus Computer Hours for Seven Large-Scale Programs.

computer hour is required for approximately every 53 instructions.

Another hypothesis resulting from the same data is illustrated in Figure 3. This shows a near-linear relationship between the two resources of manpower and computer time. It is likely that better information for predicting the number of computer hours required for the production of programs of various sizes and complexities is available in most programming organizations although it is often buried in accounting data.

A second important factor in this category is the extent of support program availability, reliability and documentation, including utility programs, debugging programs and library routines. Clearly, the more support programs that have to be produced "from scratch," the more manpower and computer time will be required for the total program development.

One other factor that may have a considerable influence on cost is the number and types of documentation produced for various types of programs. The graph in Figure 4 is based upon an analysis of five large-scale programming efforts. It suggests that there is a linear relationship between the number of pages of documentation actually produced to satisfy contract requirements and the number of instructions in the program. These data represent

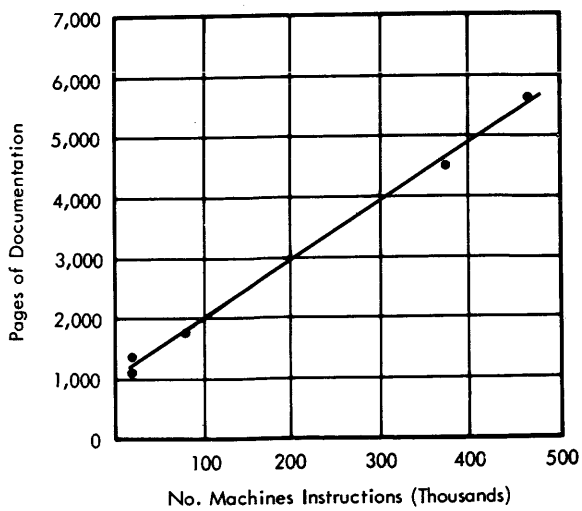


Figure 4. Pages of Documentation Versus Program Length for Five Large-Scale Programs.

documents delivered to customers. There may be many times this number of pages actually written in a large programming effort.

Some other guidelines for estimating the costs of documentation have been suggested. A drafting rate of 3 to 5 pages per day (750-1250 words) is a good rule of thumb for various types of programming documentation. A technical review rate of 20 pages per day seems average, but reviews that do not require extensive rewrites may run to 50 or 100 pages per day. Estimates on typing rates, illustrating rates and duplicating rates are usually fairly easy to obtain in most organizations.

RESOURCES THAT ARE AVAILABLE

3. *Data Processing Equipment*

The development of computer hardware is proceeding at such a rapid rate that it is difficult to make long-range estimates (beyond a few years) concerning its effect upon programming. Such improvements as faster add times, greatly increased memory capacities and speeds, multi-processors and new input/output devices may profoundly affect the accuracy of programming-cost estimation.

With respect to equipment, a critical factor is the number of hours per day that the computer is available to the programming staff. A commonly held intuitive notion is that the more hours per day the computer is available to programmers, the lower the over-all cost of the programming effort. Among the considerations in determining the number of hours per day of availability are the number of shifts per day, the time required for preventive maintenance and the number of other computer users sharing the equipment. Of course, another strong influence is computer capability; for example, large memory capacities seem to make programming easier. Further considerations are the power of the order code, the speed of access to primary and secondary memories, operating time and speed of input/output gear.

4. *Programming Personnel*

The most important factor in this category relates to the experience of the programmers assigned to the job. There are three particularly important types of experience:

(a) Experience with the particular computer—Clearly the more experience a man has with the particular machine for which the program system must be designed, the more apt he is to be familiar with its capabilities and therefore the less time it should take him to do the programming and testing.

(b) Experience with the particular language —Programming languages differ in their suitability for various types of programming efforts. Familiarity with an appropriate special language, such as NELIAC or JOVIAL, certainly makes the programmer more efficient and requires a smaller number of man months.

(c) Experience with the particular application—If the programmers have experience with the particular type of system application and/or design being programmed, then less time should be needed for the early analysis and definition phases of the job.

In addition to experience, cost is affected by the number of man months of programmer training required for the project. A possible hypothesis is that the cost of programmer training may take the form of a U-shaped curve. That is, for any given task there seems to be an optimum amount of training that the programmer should have. More training presumably would not produce a commensurate return, while less would lead to errors and confusion in the programming.

The number of people to assign to a given function or task also has an important impact on cost. There is probably an optimum number, for each type of programming effort, although research has not been conducted to determine what this is. In the field of research and development, which has some similarities with programming, the optimum number in a work group seems to be between 4 to 7.[5]

There are also costs associated with obtaining personnel—either in hiring or in transferring from other contracts and in relocating them, if necessary. This is a function of both

employee turnover and the size and type of the project. A study is currently being sponsored by the Navy at the University of Southern California to analyze the job of the computer programmer, develop criterion measures of performance and determine optimal personnel selection and classification procedures.[9]

NATURE OF THE WORKING ENVIRONMENT

5. *Management Procedures*

The design and institution of clear-cut management procedures may seem costly at the outset, but their true value must be determined by comparing the cost of the plan with the cost of not having the plan. Both these costs are extremely difficult to determine but experience in many programming efforts indicates that well planned projects enjoy higher productivity rates.

The most important effect upon total cost in this category is the use, maintenance, and monitoring of a management plan that includes at least communications and decision-making procedures, mechanisms for handling changes in the program, delineations of responsibility and authority, and schedules for major milestones and products. The plans should also specify standards for flow charts and documentation and quality control procedures. There is probably some optimum per cent of time that should be spent in planning for program development.

Another important cost factor is the number of computer runs permitted to each programmer each day. One of the reasons for the current activity in the development of time-sharing procedures is the belief that a larger number of computer runs per day for each programmer will shorten lead times and decrease costs. An equally reasonable hypothesis is that a cost trade-off exists between desk checking and computer testing such that some optimum number of computer runs will minimize total cost, the optimum depending upon the relative cost of computer time and programmer time.

6. *Development Environment*

A particularly important problem is the number of agencies with which the programming staff must coordinate. In addition to the user or customer, there may be separate agencies responsible for the contracting and for the other aspects of the system, such as hardware development. Problems of coordination and concurrence multiply as a function of the number of groups with which the programming staff must deal. Often, mutual education between these various staffs becomes necessary and the cost of briefings and meetings for such purposes may make the cost per instruction higher for large-scale program systems than for smaller ones.

7. *Facilities, Services, and Supplies*

In most accounting systems, normal overhead and miscellaneous supplies are covered by a percentage addition to the estimated direct labor and materials. However, unusual expenditures associated with large-scale programming efforts may not be adequately covered by the average overhead burden rate. For example, various types of technical and administrative support are needed to assist the programmer. Computer operators and EAM personnel save the programmer considerable time during testing. Effective and experienced management and administrative personnel assure that the work is efficiently organized and free the programmer to concentrate on technical matters. Technical editors help to ensure that documentation is adequate and understandable. There is some optimum mix of such support personnel that will ensure minimum cost.

Another important factor in this category is travel and communication cost. Trips may be required for briefings and conferences with user organizations and associated developmental agencies; for data gathering; for training and familiarization; and for concurrence on requirements or design. There is a strong tendency to underestimate these costs or to curtail them as an economic or political measure; this often results in delays in getting data or concurring on programming details, and these delays may be very costly in terms of time and manpower.

In some cases an important consideration is the cost of special facilities, e.g., simulation devices, special office equipment, or computer

room facilities. Special wiring and air conditioning, false flooring, space for storage and movements of parts and equipment, maintenance and test hardware, etc., all contribute to the cost.

SUMMARY AND CONCLUSION

In this paper we have listed some cost factors that represent a consolidation and skimming of a larger list of factors that contribute to the cost of computer programming efforts. In many cases, the factors are very difficult to measure or quantify; further, their effect upon other factors and upon the total costs of the programming effort is often difficult to determine. For many of the factors, some data exist in current accounting records, but these data have not been collected, compiled and analyzed.

The mere listing of cost factors in programming is only a first step, albeit an important one, toward the development of a more scientific and, hopefully, a more precise method of estimating the cost of programming efforts. Ultimately, one would hope to discover by analysis of data some predictors that would enable a more accurate estimate of the number of instructions, man months and machine hours. Among the cost data—both estimated and real—that we would hope programming managers would begin to accumulate for the various types of products and activities in their projects are the following:

1. The number of machine instructions and the programming language used. Also, the percentage of the finished program consisting of library routines and subprograms from previous programs.
2. The number of man months of programmer effort, including the first level of supervision, and the experience level of the programmers.
3. The number of hours of machine time required for testing and debugging purposes and the types of machines used. Also, the pattern of machine usage in terms of runs per day and hours per run.
4. The number, types, and timing of important program changes and, in at least a qualitative sense, the effects of these changes on the final product.
5. The types and number of pages of documentation required, including a notation as to whether they are single or double spaced.

In addition, it would be useful if a log could be kept by a project "historian" describing certain qualitative attributes such as those described earlier in this paper. This section should describe the data-processing functions of the program system and its relationship to other systems. It should also identify all interim and end products, such as types, listings and descriptive documents.

A series of experiments to study those factors that can be analyzed only in controlled environments ideally should be conducted simultaneously with the collection of new and existing data on program costs. For example, well designed, statistical experiments would be useful for determining the effects of different types of programming languages upon total cost, the effects of greater or lesser machine availability upon costs, the optimum size of programming staff for different types of programs, and the best mix of programming talent (experienced versus inexperienced) for given types of jobs.

Obviously, it may be quite some time before a valid predictive set of equations can be developed for program cost estimation. Nevertheless, it is to be hoped that each small step in this direction will represent a useful experience in itself in terms of increased insight into the programming process. Certainly, research into techniques for improved programming management should be encouraged if the industry is to keep pace with the increasing demands for its services.

APPENDIX I LIST OF COMPUTER PROGRAMMING COST FACTORS

Summarized below for the convenience of the reader is the complete list of cost factors discussed in this paper.

Operational Requirements and Design

1. Extent of innovation in the system, its components, and especially the automatic data-processing function.

2. Extent to which the programming contractor will participate in a determination of the information processing needs (i.e., the system and operations analysis, and the system and operational design).
3. Number, size, frequency, and timing of system design changes.
4. Extent of command and control system decentralization and number of interfaces.
5. Number of other components and subsystems being developed concurrently as part of the command and control system (e.g., intelligence, sensor, etc.).

Program Design and Production

1. Number of computer program instructions and the types of programs that must be produced.
2. Number, types, and frequency of inputs and outputs to the computer(s).
3. Extent of innovation required in the program system; that is, the degree to which programs are similar in nature to those previously written.
4. Number, types, and quality of publications and documentation for both customer and internal use.
5. Extent of complexity of the data-processing functions.
6. Degree to which the following program design characteristics are recognized and must be incorporated.
   (a) Maintainability—the ease with which program errors can be detected and corrected.
   (b) Changeability—the ease with which new functions can be incorporated in the program.
   (c) Usability—the ease with which personnel other than designers can use the program.
   (d) Flexibility—the ease with which the program can be used for other purposes with only slight modification (e.g., SAGE programs for air traffic control).
   (e) Generality—the ease with which a program can accept a wide range of inputs.
7. Extent of the constraints on program design (e.g., real-time requiremnets).
8. Number, size, frequency, and timing of program design changes.
9. Extent to which data for data base is available, or data collection is required.
10. Number of entries (total size) for the data base, the number of different types of data needed for it, and the extent to which each can serve many programs or subprograms.
11. Efficiency of the programming language and the compiler or assembler.
12. Extent of the completeness and clarity of the system test and acceptance test requirements.

Data Processing Equipment

1. Number of hours per day of computer availability.
2. Extent of capability of the computer and its suitability for the job to be done.
3. Extent to which the operation of the computer and peripheral equipment is reliable, well tested, and well documented.
4. Number of equipment components being developed concurrently with the program.
5. Number of different computers for which programs are being prepared.
6. Number and types of displays used.
7. Extent to which adequate EAM support will be available.
8. Extent to which routine preventive and emergency maintenance will be available.

Programming Personnel

1. Types and quality of programmers.
2. Number of man months of programmer training required.
3. Number of programmers to be assigned to a given function or task.
4. Policy of obtaining and phasing of personnel to staff a new contract.
5. Rate of turnover.

Management Procedures

1. Extent of use, maintenance, and monitoring of effective management plans within

both the customer's and program developer's organizations.

2. Extent of formalized procedures to use the computer facility.
3. Extent to which there is a well defined and controlled system change procedure.
4. Extent of an error-reporting and -correcting procedure.
5. Extent of contingency plans in the event the computer is overloaded or otherwise unavailable.
6. Extent of quality control that is exercised during testing (e.g., reliability requirements).

Development Environment

1. Number of agencies the programmer contractor must deal with and their level of experience with system development.
2. Average number of days and effort required for concurrence.
3. Travel requirements.
4. Extent to which delivery dates for required programming tools are reliable, and correspondingly, the amount of pressure caused by a tight schedule.
5. Extent to which the computer is operated by another agency.

Facilities, Services, and Supplies

1. Number of computer operators and EAM personnel required.
2. Number and experience of technical management personnel, administrative personnel, and technical editors.
3. Cost of special simulation facilities, computer room facilities or special office equipment.
4. Number of square feet of new office space or building required.
5. Exceptional costs of graphic arts and reproduction.
6. Cost of punched cards, magnetic tape and other special supplies or equipment.
7. Cost of special security requirements (e.g., Top Secret vault).

BIBLIOGRAPHY

1. BLEIER, R. E. *Frequency Analysis of Machine Instructions in Computer Program Systems,* TM-1603, System Development Corporation, 19 November 1963.
2. FARR, L. *A Description of the Computer Program Implementation Process,* TM-1021/002/00, System Development Corporation, 25 February 1963.
3. HEINZE, K. P., N. CLAUSSEN and V. LABOLLE. *Management of Computer Programming for Command and Control Systems, A Survey,* TM-903/000/02, System Development Corporation, 8 May 1963.
4. JAMES, J. D. W. "Measuring the Profitability of a Computer System," *The Computer Journal,* 5:4, January 1963, pp. 284-293.
5. JOHNSON, ELLIS A. "A Proposal for Strengthening U.S. Technology," in Burton V. Dean (editor), *Operations Research in Research and Development,* New York, John Wiley and Sons, Inc., p. 34.
6. LABOLLE, V. *Management Aspects of Computer Programming for Command and Control Systems,* SP-1000/000/02, System Development Corporation, 5 February 1963.
7. NANUS, B. *Cost Estimation Bibliography,* TM-1430, System Development Corporation, 20 August 1963.
8. New York Controller's Institute Research Foundation. *Business Experience with Electronic Computers,* 1959, p. 48.
9. RIGNEY, J. W., R. M. BERGER and A. GERSHON. *Computer Personnel Selection and Criterion Development: I. The Research Plans,* Los Angeles, Department of Psychology of the University of Southern California, February 1963.

HYBRID COMPUTATION... WHAT IS IT?... WHO NEEDS IT?...

Thos. D. Truitt
Electronic Associates, Inc.
Research and Computation Division
Princeton, New Jersey

INTRODUCTION

Some Historical Notes

During the nineteen-fifties the capabilities of electronic computers expanded so fast as to stay well ahead of the needs of the average computer user. Such was the case in both the analog computer and scientific digital computer fields. One effect of this situation was the formation of two schools of experts with opposite views on the choice of the "best" general purpose scientific computer. Differences of training, experience, and semantics led to a serious barrier to communications between these two groups. At best, the fashionable, middle-of-the-road position was to admit that each computing technique "had its place," which did little to break down the barrier. Only with the appearance of a computational task that could not be accomplished satisfactorily by either type of general purpose computer—only then was the barrier cracked and a small opening made.

In the aircraft and the budding aerospace industries the analog computer had long been the primary, if not the only, means employed for simulation of new or proposed engineering designs. In fact the analog computer grew up as *the* aircraft simulation tool. A mathematical model of each new design was formulated and an electronic analog model formed on the computer. Performance data obtained from wind tunnel tests were added to complete the model, and large numbers of experimental "flights were flown" as the effects upon the dynamic behavior of the model, with changes in design parameters, were examined. The characteristics and performance of the simulation model often provided input data for a digital computer program that would evaluate, optimize, and calculate final design specifications. The digital computer was also useful as a simulation tool in calculating orbits and ballistic trajectories and in designing guidance and control systems for such aerospace flights.

While conventional computer simulation techniques were quite adequate for most aerospace design problems, the exceptional costs and hazards involved in missile and manned vehicle missions required computer simulation of complete missions. Some full mission simulations required inclusion of actual control mechanisms and of a manned cockpit, thereby dictating simulation on a real-time scale. It was apparent that such total system simulations exceeded the capabilities of any single type of computer. Only by the combined application of analog and digital computers could adequate simulation be accomplished. It was hoped that the special features of the two computer types could be combined to form a hybrid simulator of superior performance.

Thus the earliest attempts to combine the computation of analog and digital computers took place in about 1958 at the Convair Astronautics plant in San Diego and at the Space

Technology Laboratories in Los Angles.[19,21,39] In both cases the job at hand was the complete mission simulation of the trajectory of a long range missile. The speed of the analog computer was a necessary element in the study to permit a "real time" simulation of the rapid motion of the vehicle and of control surfaces. However, the dynamic range required of the simulation was in excess of that of the best analog computers. That is, the ratio of the total range of the trajectory to the required terminal phase resolution (a dynamic range of 10^5 to 10^7) was greater than 10^4, the upper limit of analog computer dynamic range for small programs. Hence the digital computer was used to calculate those variables for which such dynamic range was necessary. The most important of these were the navigational coordinates—the digital computer performed the open integration of velocities to determine the vehicle's position, plus the dynamic pressure, a function that is very sensitive to altitude and velocity.

It is fortunate that in such long range aerospace trajectory simulations the variables with wide dynamic range requiring precise calculation are not, at the same time, rapidly changing. Moreover the "high speed" variables do not require precise calculation. The early combined computer systems employed the largest and fastest digital computers available at the time—Univac 1103A and IBM 704—together with 300 to 400 amplifiers of general purpose PACE analog equipment. In both cases even these fast digital computers were only just fast enough to perform the required repetitive calculations for the slowly changing variables of the simulation in real time.

Since the installations of the first combined computers at least a dozen computer laboratories have employed general purpose analog and digital computers together to solve simulation problems, and a number of attempts have been made to devise spacial purpose systems of analog and digital devices. Among the latter are the CADDA of the National Bureau of Standards and the "pulsed analog computer" of the MIT Electronic Systems Laboratory.[10,11,12,13,35,42] Hybrid computers of a unique type are the combinations of a general purpose digital computer and a digital differential analyzer (dda), illustrated by the Bendix G15 with the DA-1 attachment and the Packard Bell PB250 with the Trice dda.[27] The former system consists of a small, slow computer with an even smaller serial dda ($50,000 and $10,000 respectively). In contrast to this the Packard Bell system combines a small, medium speed computer ($40,000) with a large serial-parallel dda ($500,000). Among the systems of general purpose computers, generally large analog computers have been combined with both large (IBM 7090) and small digital computers (PB 250, LGP 30).[2,3,7,17,19,21]

A brief analysis of the applications to which existing installations of combined systems have been applied leads to these generalizations. For the most part the analog computers in these systems have been employed in a normal manner to simulate the dynamic behavior of physical systems by solving sets of non-linear, ordinary differential equations, while the digital computer has performed one or more of the following three functions: complex control logic, storage of arbitrary functions or sampled analog functions, and high precision arithmetic primarily for numerical integration. Examples of the applications are:

a. *Analog computer plus digital control logic.*

A system that in itself contains discrete control functions of continuous dynamic variables is appropriately simulated by a hybrid computer. The kinetics of a chemical process are simulated by continuous analog means while its digital control system is represented by a digital program.[29,41] Similarly an aerospace vehicle with an on-board digital computer, control system, or autopilot is simulated by hybrid techniques.

b. *Analog computer plus digital memory.*

A very common difficulty in the simulation of a chemical or nuclear reactor is providing an adequate representation of the transport of fluid in pipes from one point to another — from reactor to heat exchanger. The simulation of this transport delay of a dynamic variable, such as the time variation of the fluid temperature, is very nicely accomplished by the use of a digital computer for storage of the temperature function for a fixed, or variable, length of time.

Digital computer memory has also been used effectively to store multivariable arbitrary functions — an operation which is seriously limited in the analog computer.

c. *Analog computer plus digital arithmetic.*

This type of application is the "classic" one where the digital computer is used to perform precise, numerical integration of space vehicle velocities to keep track of the exact position of the vehicle over a very long range flight.

It should be noted that a significant difference is apparent in the applications of computer systems employing a very small digital computer and those with large, very fast computers. In general the small machines are limited to execution of control logic programs, one or two channels of transport delay simulation, or limited function generation programs. Since numerical integration and complex function generation by digital programs require considerable time for each calculation, for each discrete step in time, only the fastest digital machines can be used effectively for these tasks.

SOME COMMENTS ON THE EVOLUTION OF HYBRID SIMULATION

The term "computer simulation" appears in so many contexts it is important to emphasize that its use in these pages is limited to the modelling of complex physical systems for the study of their dynamic behavior. These systems are represented by sets of differential equations, algebraic equations, and logic equations. As in most simulation studies the objectives of hybrid simulation may be experimental design, prediction and control, and design evaluation, verification, or optimization. It is not expected that the important applications of hybrid simulation will include: data processing system simulation; information handling simulation; business system simulations; simulation of television coding, character reading machines, communications coding systems, etc. . . . These are all digital computer simulation applications. Similarly hybrid computers are not warranted for the simulation of circuits, devices, and systems for which the analog computer is quite adequate. It is in the simulation of total systems that bring together components, some of which are suited for digital and some for analog simulation, that the newer hybrid techniques are required. There probably are few, if any, simple hybrid computer applications.

If hybrid computation can be said to be a field of endeavor it must be considered to be in the formative stage. Developments to date have led to equipment configurations and programming techniques that were dictated by specific problems and limited objectives. The growth rate of the field will be determined by the extent to which a broader view is taken of hybrid computer programming techniques and applications. The greatest advances in computers have been made when the experiences of users, programmers, have been brought to bear on design of equipment. For the most part, hybrid computers of today consist of general purpose analog and digital machines, which are *not* designed for hybrid operation, tied together by "linkage equipment" designed only to solve the communication problem. This has been a necessary first step. Newer hybrid systems will be designed not just for communications but for efficient solution of hybrid problems and for convenient programming. The purpose of the following discussions is to bring attention to the essentials of hybrid computation and their relationship to problem solutions. Out of this should come some indication of how present day computers can be most effectively employed. The next generation of hybrid computers will undoubtedly achieve a greater degree of integration of parallel computing elements with the sequential stored program principle. Eventually patch panel programming of parallel elements will be replaced by an automatic system, thus affording a fully automatic method for computer set-up and check-out. Even today a secondary activity of the digital part of the hybrid machine is the partial automation of set-up and check-out of the analog computer. This feature becomes increasingly important as the computer system grows in size and the programs grow in complexity, for the attention of the programmer and problem analyst needs to be directed to the simulation itself rather than the simulator.

THE ELEMENTS OF HYBRID COMPUTERS

Digital Computers

Many conflicting factors have influenced the choice of digital computers used in hybrid sys-

tems. Computer speed and economics have probably been dominant. Since there are so many computers on the market today that have sufficient speed and that span the complete range of prices, it is more instructive to examine the features that are essential for hybrid computation.

Speed. The speed of execution of arithmetic operations is most important, and this is a function of memory access time and multiplier speed. The access speed of drum and delay line memories is too slow. Magnetic core access times of 2 to 5 microseconds are currently popular. This means the time for addition of two numbers is 4 to 10 microseconds. Multiplication and division take longer—times of 15 to 40 microseconds are generally available and quite satisfactory. Overall program speed can be increased by the use of index registers—three registers is desirable; more are useful. Special instructions for subroutine entry, for executing commands out of sequence, and for testing and skipping can help increase computing speeds.

Word Structure. The basic requirement is for a fixed point, binary word of at least 24 bits. Since round-off errors affect the last several bits a smaller word size would result in a dynamic range limitation of less than 10^6. A longer word may be useful in a few applications where fixed point scaling may be difficult. Floating point computations may make things easier for the programmer but should not be depended upon at the expense of computational speed. It may be noted that the equivalent of fixed point scaling is a necessary part of the analog program, and hence floating point operations may not prove as advantageous as for some all-digital programs. Decimal format and character oriented machines do not offer any advantages for hybrid computation, and usually they are slower than equivalent sized binary computers.

Input/Output. High data rates in and out of core memory and any feature that minimizes loss of computing time for input/output operations are highly desirable. In addition a fast, flexible means for communicating control signals to and from the analog section of the hybrid system is necessary. Three kinds of control signals are usually provided: interrupt and sense lines, and output control signals. It is by means of these controls that the sequential operations of the digital machine are made compatible with the parallel, simultaneous operations of the analog machine. Since communications must be made with many points in the analog computer, a number of these control signals are needed. Interrupt signals, from outside the computer, stop the current sequence of calculations and force transfer to another sequence. Sense lines simply indicate to the digital program the current state of devices outside the computer. They may be sensed by specific programmed instruction. Other programmed instructions will send control signals outside the computer on the output control lines.

Memory. As noted above the digital computer main memory should be a high-speed magnetic core. Since most hybrid applications do not require a large memory for either program instructions or data, four, eight or twelve thousands words of core memory should suffice. Larger memories may be desired for special digital programs and larger hybrid problems when more experience has been gained in this field, thus expandability of a memory to 16K words is a good feature. Newer computers are being introduced with small, very high speed "scratch pad" memories. Such memories may have cycle times less than a microsecond and are used to store intermediate arithmetic results. This feature increases the overall computation speed of the computer.

The normal manner of operating an analog computer involves a fair amount of non-computing time when the computer remains in the Hold or Reset mode. These intervals may range from seconds to minutes while adjustments are made, pots are set, or recorders changed, or while the programmer analyzes results. It is not possible for the analog computer to operate on other programs at these times, however, with a hybrid system, where such waiting periods are likely to occur also, it is reasonable to consider having the digital computer work on a different program during the intervals, whatever their length. Appropriate "interrupt" and "memory lock-out" features are possible to permit time sharing of the digital machine without affecting the hybrid program and without the danger of one program interfering with

the other. The secondary program (a strictly digital problem) is simply stored in a "protected" part of the core memory and utilizes all the bits of time not required by the hybrid program.

Peripheral Equipment. In many digital computer installations the investment in peripheral equipment rivals that in the central computer. Current hybrid computer applications require only a minimum of digital peripheral equipment. The graphic output equipment associated with the analog computer is sufficient for computational results. Punched paper tape reader and punch and typewriter may be all that is required for programming. Larger systems in the future will employ punched cards and magnetic tape, primarily for rapid change-over of problem and automatic check-out. Large off-line data storage does not appear necessary for most applications.

In summary the digital computer must be characterized as a *sequential* machine. For effective use within a hybrid system the machine (a) must have sufficiently high internal speed for it to *appear* as though a number of calculations were taking place simultaneously; (b) must be organized for maximum speed in executing mathematical calculations; and (c) must have efficient means for input and output of data *during* calculation (Figure 1).

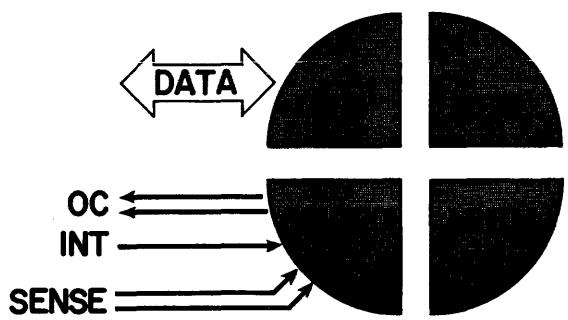


Figure 1. The digital computer must be characterized as a sequential machine. For effective use within a hybrid system the machine must have sufficiently high internal speed for it to appear as though a number of calculations were taking place simultaneously. It must be organized for maximum speed in executing mathematical calculations; and it must have efficient means for input and output of data during calculation.

Analog Computers

In contrast to the above the analog computer is a *parallel* machine, with many computing components and I/O devices operating in concert. There are few, if any, features of the modern analog computer that are not appropriate to a hybrid system. However, only the largest analog machines have been used for general purpose hybrid simulation. The common measure of a large computer is that it has 100 to 200 operational amplifiers. Since two or more computers may be "slaved" together, larger systems are possible when required.

Analog computer features that are important for hybrid systems can be simply listed as:

—Integrators with multiple time scales

—Amplifiers for tracking and storing voltages

—Very fast control of the modes of individual amplifiers

—Automatic, remote control of the setting of potentiometers

—Fast, accurate multipliers and trigonometric resolvers

—High speed comparators with logic signal outputs

—Electronic switches (logic signal gating of analog signals)

In the early days logic equations or switching functions were programmed with relays and stepping switches, which were connected to the patch board by various means. Present day technology employs electronic switching of integrator modes and voltage signals at high speeds, and the delays inherent in relay devices can no longer be tolerated for logic operations. The logic building blocks common to the digital computer designer (flip-flops, gates, inverters, monostable multivibrators, shift registers, and counters) are ideally suited to these operations. Thus with electronic switches replacing relay contacts, logic modules have become an integral part of all new, large, analog systems. These modules are programmed like the other analog components by interconnections at a patch panel. Many signals occur simultaneously but they are logic signals—two values, Zero and One, that change as functions of time. Input signals to logic programs come from comparators, manual switches, and external control sig-

nals. Logic program outputs go to integrator mode controls, storage amplifier controls, electronic switches (DA switches) to gate analog signals. As will be shown later, it is essential for a hybrid system to have a very significant complement of digital logic components. A few gates and flip-flops are not sufficient. The potentialities for use of logic components in an analog computer for hybrid operation are so great that the EAI HYDAC Digital Operations System is an entire computer console with its own patching system used entirely for the programming of digital components for parallel computation. This console is really a complete *logic computer*. It is used together with a conventional analog computer to form what is truly an all parallel hybrid computer.[1, 9, 23, 24, 29, 40, 41]

In summary the modern analog computer must be characterized as a *parallel* machine. It is not solely a computer for continuous variables. It is a parallel assemblage of building blocks: integrators, multipliers, etc., for continuous variables; and flip-flops, gates, counters, etc., and "digital" circuits for discrete variables. It is organized for convenient representation of an "analogous" physical system by means of a computer model constructed of these building block (Figure 2).

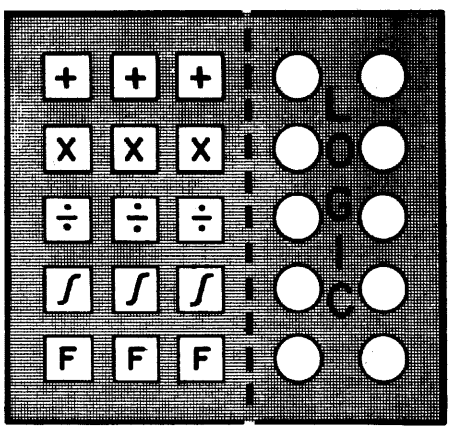


Figure 2. The modern analog computer must be characterized as a parallel machine. Not solely a computer for continuous variables, it is a parallel assemblage of building blocks: integrators, multipliers, etc, for continuous variables; and flip-flops, gates, counters, etc., and "digital" circuits for discrete variables. It is organized for convenient simulation of an "analogous" physical system by a computer model that is constructed of these building blocks.

Conversion Devices

In providing data communication between a sequential computer and a parallel computer three kinds of devices are commonly used: the multiplexer, the analog-to-digital converter (ADC), and the digital-to-analog converter (DAC). In addition, all early systems have employed a timing and control unit which performs a relatively fixed set of operations, with manual switches to select options such as sampling frequency, and number of channels. Such "linkage systems" thus consisted of a timer unit plus a group of linkage building blocks prewired to perform a specific task. With the integration of digital logic components into the parallel computer, however, greater programming flexibility is possible by use of these logic units for timing control of the data conversions. Furthermore, the converters and multiplexer can act very naturally as additional building blocks in the parallel computer. Thus it is likely that future hybrid systems will simply incorporate the "linkage system" within the parallel computer.

Usually several or many analog signals in a hybrid program will be sampled, converted and transmitted periodically to the sequential digital program. The numbers, of the several sequences of numbers to be entered into the core memory, can be accepted only one at a time. Since this is so, the conversions from voltage to number form can be performed one at a time—first from one analog variable and then from another. The multiplexer is used to select one from many analog signals, to step through a sequence of signals, and thus to furnish voltage input signals to the ADC.

The output of commonly used ADC's is a binary number of 10 to 14 bits. A 13 bit binary output probably is the best compromise; it represents a resolution of one part in eight thousand, and resolution of analog voltage signals is at best one part in ten thousand. Conversion times range from 50 to 300 microseconds. A typical time of 100 microseconds would allow the converter to be shared by 16 analog signals each with a frequency spectrum extending to 20 or 30 cycles per second. This will be explained later.

DAC units should have the same binary word size as the ADC, except for special low accuracy

uses. Conversion times are not determined by the converter but rather by the bandwidth of the analog amplifier following the converter. Data from the sequential computer appear only one word at a time, and some means for retaining the latest value, of each sequence of numbers, for each output function is needed. The sequence of numbers coming from the computer may first be converted to voltage values by a single DAC, and then distributed to storage amplifiers for each channel. It is more customary, however, to hold each of the latest words for each channel in a digital register which is an integral part of the DAC assigned to each channel (Figure 3).

Special Forms

As a passing thought it may be noted that while the primary emphasis here is being placed on the distinction between parallel and sequential operation, the term "hybrid," historically, has been used to imply the combination of continuous and discrete calculations, and that therefore consideration might be given to two special kinds of hybrid computers:

The Parallel Hybrid Computer, which is a proper term for the EAI HYDAC 2000 machine. This system combines an analog computer with a general purpose system of programmable logic building blocks, multiplexer, ADC, DAC's,

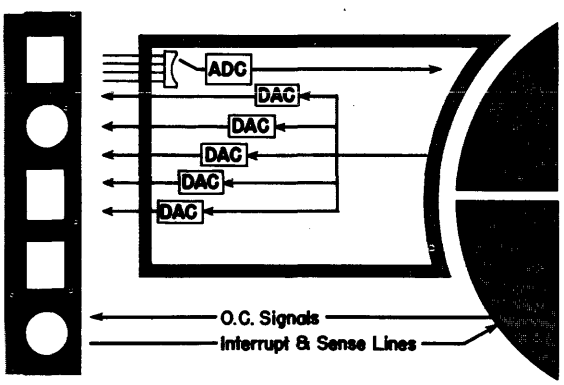


Figure 3. ADC's and DAC's are format converters, changing voltage to numbers and vice versa. As components of the parallel machine the converters together with logic components must act to "match impedances," i.e., resolve the incompatibility between the parallel and sequential programs.

digital memory units for storage of sampled analog functions, and several digital numerical adders and subtractors. The application of this system encompasses an intermediate range of hybrid problems, such as:

a. Transport delay simulation[24]
b. Single and multi-variable function generation
c. Logic control systems[23]
d. Automation of the analog computer for parameter searches and optimization studies[41]
e. Simulation of numerical and sampled data control systems[1, 9]

The Sequential Hybrid Computer, which is exemplified by the experimental "pulsed analog computer" techniques developed at M.I.T. for use in an aircraft flight trainer.[10] This system employs a sequential digital computer which controls a small number of analog functional components—one multiplier, one reciprocal generator, one integrator, and several adding units. These units are interconnected and receive inputs by digital program control. They form, in effect, "analog subroutines" for the sequential computer.

THE SEQUENTIAL/PARALLEL HYBRID COMPUTER

The term "hybrid" is most appropriately used to indicate the combined use of sequential and parallel computing techniques, first because the future growth in hybrid simulation will be predominantly in this direction. Second, and more important, is that from the standpoint of the programmer who must bring the two types of computers together to find a useful solution to a problem, the only really significant disparity lies between sequential and parallel operations. The difference between continuous voltage and discrete number is simply one of format. It would make little difference if the analog signals were frequency modulated, pulse code modulated, hydraulic, or pneumatic—appropriate format converters could be found. The feature of hybrid computation that *is* of importance is: in part of the machine many operations are taking place simultaneously, and many time-varying problem variables exist in parallel; while elsewhere a number of operations take place, one at a time in a repetitive

manner, so as to effect the generation of several problem variables, *as if* they occurred simultaneously. Furthermore, the parallel computation is tied to a real-time base: the very passage of time itself accounts for the changing of the basic independent computer variable. The sequential program is asynchronous—not controlled by a clock. Operations are executed in sequence at whatever rate is possible, and for any reference to be made to the actual elapsed time, external communication is necessary. This basic incompatibility requires that the interface between the two types of operation embody more than the simple format conversions performed by the ADC and DAC's. It is necessary for data and control information in the parallel machine to be available to the sequential machine and conversely that the latter be able to send data and control signals to many points in the former. Coincidence or simultaneity of events communicated to and controlled from the sequential program are particularly difficult to handle. The logic and data control of the interface equipment must resolve these differences in timing and operation. What might be termed an "impedance matching" device is needed between the parallel and sequential program in order to make most efficient use of both machines. The exact manner in which this is done will vary from problem to problem (Figure 4).

A Simple Example

An example will illustrate some basic considerations in defining a general purpose hybrid system. First, the operation of the simplest of linkage systems: an ADC and multiplexer, a number of DAC's, and an "interrupt clock." The flexibility of the stored program digital computer is relied on for control of these units. Assume 10 analog signals are to be converted one at a time. These words are placed in memory (average program time: 40 μsec. per word and then about 7.5 or 8.0 milliseconds of sequential, digital calculations takes place, followed by output from memory of ten words (20 μsec. per word) to ten DAC's. The entire cycle requires $7500 + 10 \times 40$ (input) $+ 10 \times 20$ (output) $=$ 8100 microseconds of digital program time. If it is assumed the conversion of the data (A to D) requires 150 microseconds per word then 1.5 additional milliseconds, or 9 ms are needed if everything proceeds sequentially. Assume further that because of the frequencies, or the analog signals, it is necessary to sample at least some, and therefore all, of the channels at 100 samples per second. A "real-time interrupt clock" is set at 100 cycles per second. This timer unit is an adjustable oscillator that sends an interrupt pulse to the digital computer. The latter then activates the ADC, waits 150 μsec for completion of a "Ready" signal, steps the multiplexer to the next channel, stores the converted word in memory, and then repeats this cycle ten times. With the tenth step the multiplexer resets to the first channel. The program then proceeds with the 7.5 milliseconds of calculation, outputs ten words, one at a time, to ten DAC's, and then waits for the next

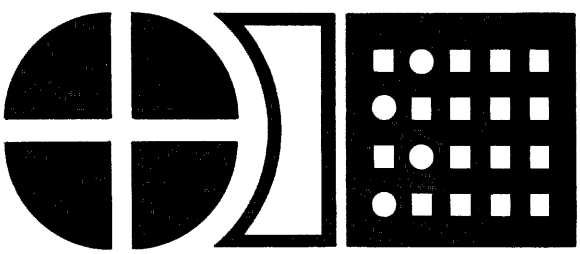


Figure 4. A hybrid computer is a compatible system of parallel computing components, both digital and analog, and a stored program sequential machine. The hybrid computer programmer must constantly be aware of the relative timing of events in the parallel and sequential parts of the program. "Matching impedances" between these parts is accomplished by programming of the interface components to suit the simulation.

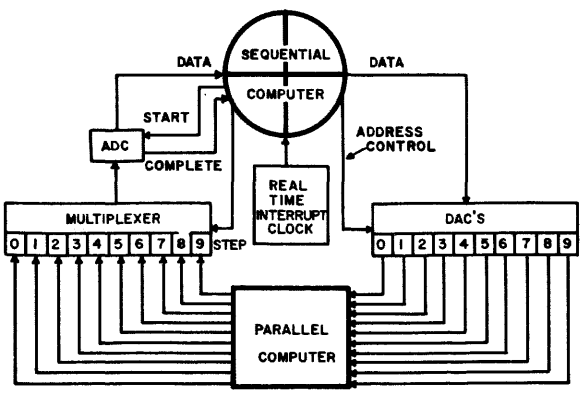


Figure 5. An early hybrid system configuration. The sequential program controls the timing of the conversion cycle. The cycle is initiated by the "interrupt clock." For a tyical problem the clock might be set for 100 cps; and 7 to 8 milliseconds per cycle would be available for calculation.

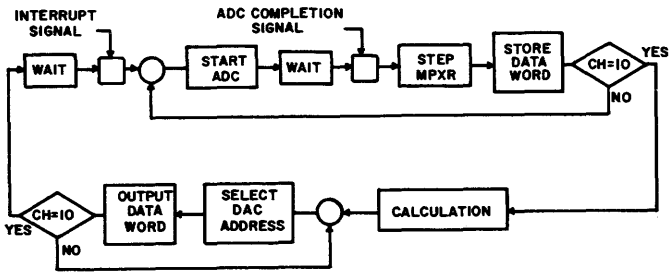


Figure 6. The sequential program flow diagram, for the example hybrid system of Figure 5, shows the steps required for control of the converters.

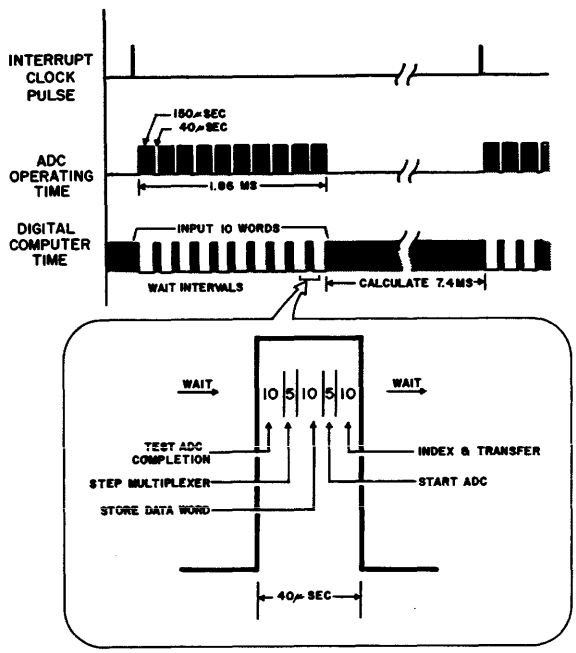


Figure 7. Typical times are shown for the steps in the conversion cycle of the example system of Figure 5. All ten channels are converted and stored in memory as fast as possible before proceeding with calculations—at the expense of intervals of "waiting time."

interrupt pulse. (See Figures 5, 6, and 7.) Manual controls are provided for selecting the interrupt clock frequency and the number of channels in the multiplexer stepping cycle.

This is certainly a simple system and it appears to satisfy the basic requirements for communications. Some of the shortcomings of the system are apparent: sampling and outputting of each channel do not take place simultaneously, 15% of the sequential program time is "waiting" time, and 3 to 5% is used to select and control devices external to the sequential computer. In other problems these percentages may be higher. The other weaknesses in this system lie in the fact that it was not designed to be a general purpose system. It is restricted in application to a class of problems for which the periodic "input/calculate/output" cycle is useful.

System Improvements or Variations

By programming the parallel digital components of the parallel computer to perform timing and control functions for the system the following changes to the above system are suggested:

a. *Simultaneous sampling.* If the sequential program operates on two or more of the input numbers together to calculate an output, then errors may occur since the input numbers were sampled at different times and correspond to different values of the independent variable. A similar effect may occur at the output since the numbers in a group of ten appear at the ten DAC's at different times. It is certainly possible by numerical means to compensate for the errors, at the penalty of additional program time.[18, 28] The common solution is to add memory to each of the ten input and ten output channels. Ten Track/Store amplifiers are added before the multiplexer and a control signal causes them all to sample, by storing the voltages, simultaneously. At the output, 13 bit registers are added in front of the DAC's. When all ten registers have been loaded, a transfer pulse causes all DAC values to be updated at the same time.

b. *External timing of ADC and multiplexer.* Sequential program time can be saved by permitting the control of the ADC, multiplexer, track store cycle to be controlled externally. A simple clock, counter, flip-flop, and group of logic gates will permit the input conversion cycle to run at its own rate—interrupting the sequential program only at the completion of a conversion. Thus the conversion time can overlap the calculation time, eliminating the waiting time. Upon interrupt only 10 to 20 μsec. may be required per sample; many control steps are eliminated. Similarly on output, the addressing and selection of output channels can be done by simple circuits rather than using sequential program time (Figure 8).

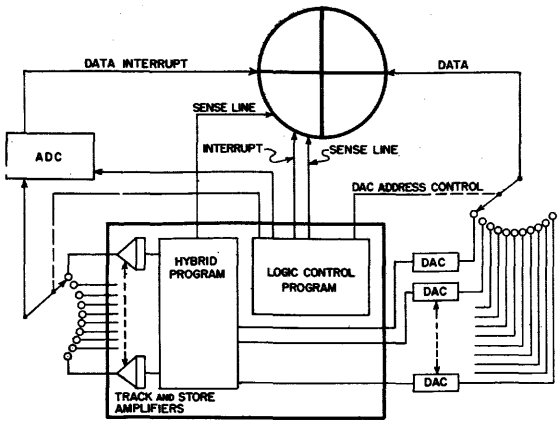


Figure 8. Several improvements to the interface system of the example of Figure 5 include: simultaneous sampling, external ADC and multiplexer timing by a parallel logic program, use of sense lines to reduce conversion channels, detection of random events, multiple sampling frequencies, and/or asynchronous sampling.

c. *Real-time clock to establish sampling frequency.* If the sequential calculation involves numerical integration over a long term, the accuracy of the sampling interval is just as important as the round-off and truncation error in the numerical calculation. Although numerical means may be resorted to for very accurate integration, in a hybrid program the calculations still need to be referred to a real-time base. This is done by using an accurately calibrated source for setting the sampling interval, or frequency. A good high-frequency crystal stabilized oscillator is an important part of the parallel digital subsystem. Sampling frequencies lower than the oscillator frequency are selected by use of preset counters.

d. *Use of sense lines to reduce number of conversions.* In the simple example problem only whole number data are transmitted to the sequential program. Thus, if the relative magnitude (greater or less than) of two analog signals is needed in the digital calculation, the two numbers must be converted, stored, and then compared. This can be accomplished more simply by use of an analog comparator the output of which is sent to the digital computer by a sense line—saving time and equipment. The state of any parallel logic component may be monitored conveniently by sense lines. These are tested in one memory cycle (2-5 μsec.). If many such communications are needed the savings will be significant. Sense lines should also be added to allow the sequential program to determine the modes of the analog computer, the relative sizes and signs of error quantities, and the states of recording devices.

e. *Detection of random events.* With fixed, periodic sampling the sequential program cannot tell exactly when events take place in the parallel machine. With comparators and parallel logic, complex functions of analog variables can be monitored. For example, it might be required to determine when the overshoot in x_1 exceeds x_2 after the third cycle, but only when x_3 is negative and x_4 is less than x_5. After determination the sequential program can be interrupted to perform specific conversions and calculations—asynchronously with respect to the primary conversion cycle. In this manner the parallel logic avoids the delays in the sequential program and uses the latter only when required. The parallel logic program analyzes the data, interrupts the sequential program, and sets up the proper channels for conversion in and out of the digital computer (Figure 9).

f. *Multiple sampling frequencies.* In the example problem all channels are sampled at a frequency determined by the highest frequency present in any one channel. It may often be the case that there are two or more groups of variables with different ranges of variable frequencies. It may then be appropriate to sample each group at different frequencies. Another ap-

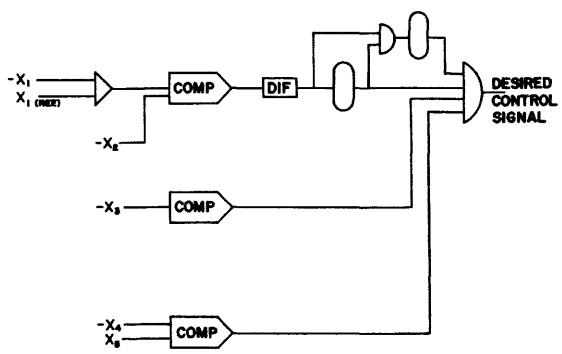


Figure 9. A parallel logic program is used to detect random events in the parallel computer. The "desired control signal" interrupts the sequential program when the overshoot in x_1 exceeds x_2 after the third oscillation; but only when x_3 is negative and x_4 is less than x_5. x_1 through x_5 are analog voltages.

proach using different sampling rates is to use several eight channel multiplexers in cascade so that the output of two of them feed two channels of a third which feeds the ADC. On each cycle of the third unit the first two are stepped, yielding different variables for those two channels on each cycle. Alternatively, each time the third steps to the two special inputs the corresponding multiplexer makes a full cycle. Timing control of these operations is performed by parallel logic components (Figure 10).

g. *Asynchronous sampling.* A completely asynchronous conversion system has been designed by one computer laboratory, in which the sequential program is interrupted only by comparators. Twenty analog problem variables are compared to reference values that are adjusted by the digital computer when necessary. Each comparator calls for conversion of some group of variables (the same variables may be called for by different comparators). When two or more comparator signals occur simultaneously or during a conversion operation, two levels of priorities are set up by logic elements to determine what interrupts are to be made. While the system appears complex, it is accomplished in a simple fashion in the parallel computer and makes good use of the sequential computer time.

A longer list of useful variations in the control and timing of sequential/parallel communications can be compiled. For the most part, however, they should be explained in terms of the particular problem applications.

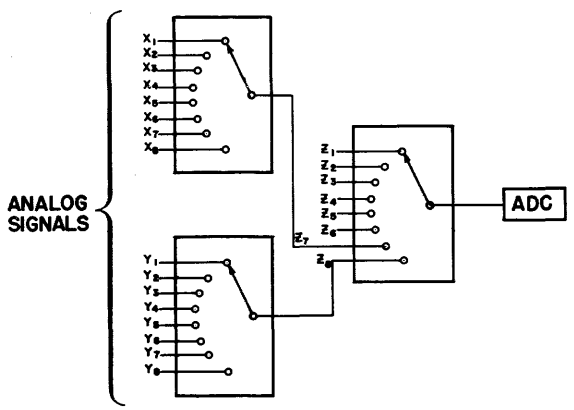


Figure 10. Cascaded multiplexers, in this or other configurations, can aid in establishing multiple sampling rates for different groups of analog signals.

Operating Times for Typical Mathematical Functions

The repeated emphasis upon the efficient utilization of the sequential program time, high arithmetic speeds, and programming tricks to gain speed can be seen to be warranted when one examines the sequential operating times for several typical mathematical functions, which, on the analog computer, would be executed continuously and in parallel.

a. The sum: $a + b + c + d$ — 40 microsec.
b. The expression: $ax + by + cz$ — 160 microsec.
c. Sinwt or coswt — 215 microsec.
d. Square root of $x^2 + y^2$ — 432 microsec.
e. Generate $z = f(x, y)$, where two dimensional interpolation is required between functional values evenly placed in x and y: — 0.5 to 1.5 millisec.
f. Rotate a vector through three coordinate angles: — 2 to 6 millisec.
g. Perform one integration of a single derivative for a single time step: — 0.1 to 1.3 millisec.

A program of three first-order differential equations, where the derivatives are calculated from the functions above (items a through f) would not be a large program; and yet for a single step in time, the calculation time would be about 11.2 milliseconds.

$$\frac{dx}{dt} = -x + y + z + f(x,y)$$

$$\frac{dy}{dt} = ax - by + cz$$

$$\frac{dz}{dt} = x^2 + y^2 + \sin zt$$

Allowing another millisecond or two for control and input/output instructions one can estimate the real-time speed performance of this program. The speed is best expressed in terms of the useful upper frequency (at full scale), in a problem variable, that can be represented by the computer. Although the example equations have no real meaning, the frequency limit for such a program is about 1½ cps. This does not seem like very fast performance for so few, simple equations. On the other hand, it is fast compared to frequencies of some of the vari-

ables in an aerospace simulation program for which digital precision is required. In any event, it should be clear for best utilization of the sequential computer, care should be taken to reduce operating time whenever possible.

Sampling Rates

When parallel and sequential machines are connected in a closed loop it is assumed that at least part of the task of the sequential program is to accept sequences of sampled values of continuous inputs, calculate functions of these inputs, which are then transferred out as sequences of numbers to be smoothed into continuous signals. The digital computer attempts to appear as if it were a parallel computer, and as in a movie projection the effectiveness of this approximation is determined by the ratio of the frame or cycle rate to the rates of change of the signals communicated, and the time resolution of the person or computer receiving the information. Thirty frames per second will not catch the information in the trajectory of a humming bird. A higher rate is needed for an accurate recording of the flight. The human eye, however, cannot resolve time intervals less than 1/30 of a second. Thus an accurate recording can be transmitted to the eye only by a time scale change to slow motion. Fortunately, the parallel computer has a time resolving power sufficient to detect the shortest practical cycle time on the sequential computer, so the limiting factor in determining useful cycle frequencies is the rates of change of the variables that pass between the two computer sections. It is customary to speak of the bandwidth or spectrum of these signals—or more particularly the highest useful frequency that must range over the full magnitude scale. The sampling rate or cycle rate must be selected in terms of the number of discrete numbers or voltage samples necessary to represent this highest frequency at the desired accuracy.

In sampled data theory the "sampling theorem" states that the sampling frequency must exceed two times the highest signal frequency if all the information in the original signal is to be retained.[36] That is, some number greater than two samples per cycle is necessary. Another important point comes from the theory: in sampling voltages at the input to the digital computer the rate must exceed twice the frequency of any signal present. If noise signals are present that are higher in frequency than the desired signal and exceeding ½ the sampling frequency it is possible for this noise to be reflected into the signal spectrum, thereby destroying useful information. This can be avoided with appropriate noise filters. If this were the only limitation it would be fortunate. However, too few samples per cycle makes it difficult for the sequential program to extrapolate and predict what takes place in between samples. It is possible to do this, of course, by numerical means, but at cost in program time. Furthermore, numerical algorithms for integration are sensitive to the ratio of sampling interval to the rates of changes of the variables, and the calculations may become unstable if too few samples per cycle are used (Figure 11).

The most important criterion for determining sampling rates appears in the conversion of the discrete data to continuous analog functions. Two sources of error affect the accuracy of the resultant continuous function. The first is the delay due to conversion and the sequential program itself. The output numbers are functions of input numbers that were sampled at an earlier time. Since the delay is unavoidable, but is predictable, numerical means are used to extrapolate the data to the time of actual digital-to-analog conversion.[18,28] The second error source is in the mechanism for conversion from

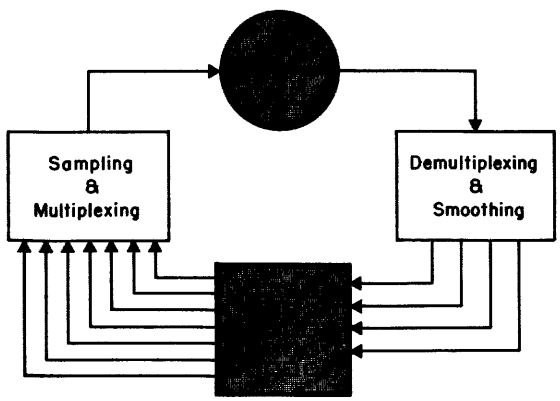


Figure 11. A Sequential-Parallel Hybrid Computer: The noise and delays due to sampling and multiplexing on the one side and the magnitude and phase errors due to demultiplexing and smoothing on the other are dominant factors in determining the proper sampling frequency.

discrete to continuous form. A sequence of discrete values fed to a DAC results in a "staircase" analog function. The desired output is a smooth curve passing through each data point (the left corner of each step). If the staircase is smoothed with an analog circuit the result is a smooth curve shifted in time $\frac{1}{2}\Delta t$, passing through the center of each step. The amplitude of this curve is attenuated from what it should be. This technique of smoothing is called "zero order" filtering. (See Figure 12.) The size of the errors is a function of the number of samples per cycle. At ten samples per cycle the magnitude attenuation is about 1.1% and the phase shift is 18 degrees. At 30 samples per cycle the errors are 0.7% and 6.5 degrees.

A first order filter may be applied to the DAC output to reduce the errors. For special purposes higher order extrapolating filters are feasible. These filters are programmed from analog components. The first order filter extrapolates from the last two discrete values to generate intermediate values until the output voltage is reset to the next discrete value from the DAC. The first order filter has a much improved phase characteristic but at low sample rates the magnitude is erroneously accentuated. The

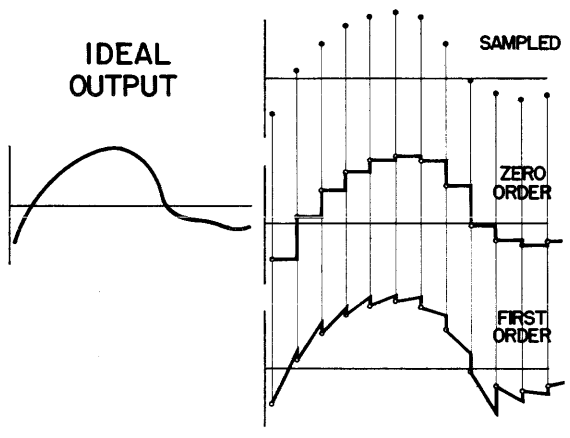


Figure 12. The discrete to continuous signal conversion requires smoothing and hence yields only an approximation to the ideal output from the sequential computer. The zero order conversion simply holds the output voltage at the last sampled value until the next arrives. When the "steps" are filtered out the result is shifted in time by the width of one half step. The first order scheme uses the past two converted values to predict the value between points, before "resetting" to the next value.

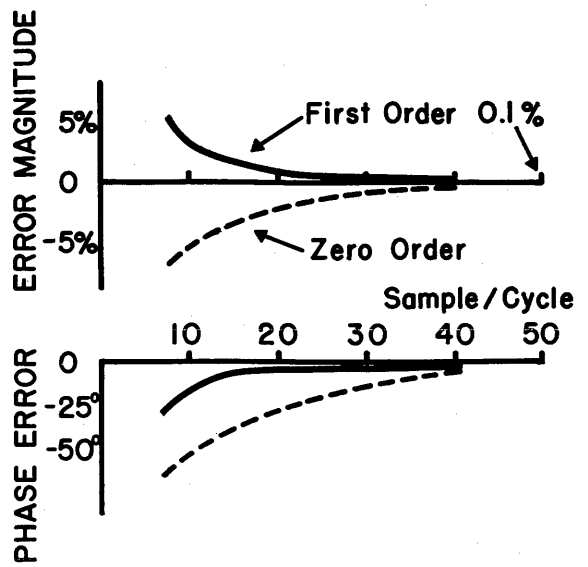


Figure 13. Zero and first order conversion methods are compared, after filtering. The errors are functions of the number of samples per cycle of full scale signal frequency. A good rule of thumb for the zero order filter is that 50 samples per cycle, the errors are less than 0.1% and 3.6 degrees.

error characteristics of the two filters are shown in Figure 13. A good rule of thumb for the simple zero order filter is that for 50 samples per cycle the errors are less than 0.1% and 3.6 degrees. The above rule is convenient for estimating the required sample rate and hence the time available for the sequential program. If the variables converted from digital vary at a maximum frequency of 2 cps, then 100 samples per second are needed, and 10 milliseconds is the cycle time for the sequential program, for primary calculations and input/output operations.

Dynamic Range of Dependent and Independent Variables

The time resolving power of the analog computer was mentioned above in connection with an analogy to the resolution of the human eye. It is instructive to pursue this concept further. The time resolving power of a computer is measured by the shortest time interval that can be accounted for in a calculation. For all signals in an analog computation, the resolution is directly related to the bandwidth of the components; however, the computer's ability to respond to on-off signals and very short pulses, or

to discriminate between two closely spaced events is a closer description of time resolution. In a digital computer the absolute minimum resolution might be taken as the time to execute three instructions; however, within a hybrid system the resolution of the sequential program is either the sample interval discussed above, or at best, for asynchronous operation, the time for a complete interrupt program to respond to an event. In the parallel computer the time resolution is, of course, much greater because computing elements need not be time-shared. For relay circuits the resolution is about 1 millisecond, for electronic switching of analog signals from 10 to 100 $u$seconds, and for parallel digital operations from 0.1 to 10 $u$seconds. If these numbers are compared to the total length of a typical computer run, say 1.5 minutes, computer time resolution can be measured by a nondimensional number:

Parallel digital logic operations	1: 10^9
Parallel digital arithmetic operations	1: 10^7
Parallel analog, electronic switching	1: 5×10^6
Sequential program, minimum useful program	1: 3×10^5
Parallel analog, relay switching	1: 10^5
Sequential program, typical sampling	1: 5×10^3

The digital computer is employed in a simulation where the dynamic range of dependent variables requires a wide dynamic range (reciprocal of resolution) in the computation. It is seen that resolution of the independent variable is traded for that of the dependent variables when a particular calculation is moved from analog to digital computer. Figure 14 shows these functions plotted against each other for different computers. The flat part, or "operating range," of the sequential computer plot is seen to be limited on one end by truncation error and the other by round-off error. This is to be interpreted as meaning that for a given set of mathematics short cuts and approximations may be used to improve the time resolution up to a point where the truncation error becomes serious. On the other end, special techniques may be used to reduce round-off error, including double precision operations, at the expense of time resolution.

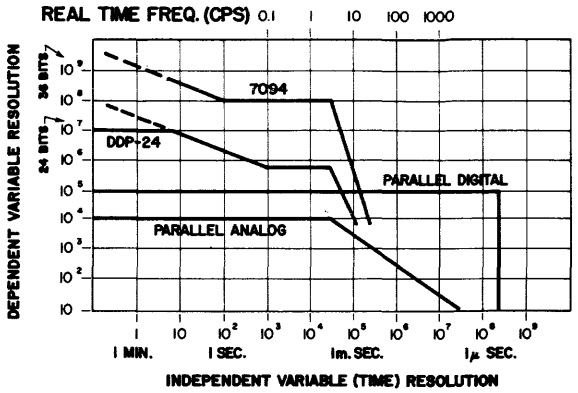


Figure 14. Certain performance characteristics of computers can be deduced from this comparison of the available resolution in the computer representation of dependent and independent variables. Notes: (1) The limits on the "Parallel Digital" curve are given as those of the EAI Hydac, 0.5 μsec. in time and 16 bits in magnitude; (2) the attenuation on the right hand end of the DDP–24 and 7094 curves corresponds to truncation errors; (3) the slope at the left of these curves corresponds to reduction of round-off errors at the expense of speed including use of double precision methods (broken lines); (4) The "Real Time Frequency" scale refers to signal frequencies passed by the sequential program (DDP–24, 7094) assuming a maximum sampling rate of 100 samples/cycle.

When the digital computer curves are related to the frequency scale, at the top of Figure 14, a particular size program must be considered. For example, at the 2 cps point, six second order differential equations for a trajectory simulation could be calculated, for that point corresponds to a 10 millisecond (bottom scale) sequential program time interval. At the 20 cps point, either very crude integration algorithms and approximations are used for the same problem, or one would be considering a much smaller calculation. Moving to the left on the curve, more time is available either for more accurate calculation or for computation of more functions. The useful operating ranges for the different techniques are evident from this figure, and this point of view should be useful to the problem analyst in considering hybrid simulation.

FORMULATION OF THE SIMULATION MODEL AND PROGRAMMING

Mathematical analysis of the behavior of physical processes and systems is a basic tool

for the design engineer, and the frequency of its use has been growing for many years at an increasing rate. At first, analysis was restricted to the smaller elements in a system, to linearized approximations, or to phenomena that can be isolated from interaction with its environment. For example, there have been many studies of noninteracting servo control loops, heat diffusion in devices of simple geometry, the "small signal behavior" of aircraft and their control systems, and batch and continuous chemical reactors of simplified geometry. Analysis starts from a consideration of the basic laws of physics as applied to the process at hand and proceeds to develop a mathematical model. The solution to the equations of this model for a range of the independent variable(s) constitutes a *simulation* of the process. The nature of the designer's task and the very fact that the analysis has been limited to an element of a more complex system, requires that many such solutions be calculated. The *simulation* is performed numerous times over to determine the variations in the process behavior with changes in (a) internal design parameters of the process, and (b) environmental conditions. Electronic computers, of both types, have aided immeasurably in reducing this task to a manageable one. The facility in obtaining simulation results that computers have afforded the designer and analyst has accelerated the general acceptance of the analytical approach to difficult design problems.

In addition to the wider use of simulation the successful correlation of experimental results with analytical predictions has built confidence in these techniques, which has led to the undertaking of simulations of more complex systems. It was once felt that a simulation model could be made so large that the analyst would have difficulty coping with the variables. Indeed this can happen, when poor engineering judgment leads to a model with many more variables than known conditions and assumptions. However, a complex model carefully built up from verified models of subsystems may lead to valuable results attainable by no other means. Thus as analysis and simulation have yielded understanding of the behavior of small systems, a natural process of escalation has led to simulation of systems of greater and greater complexity. The increased sophistication of simulation models has made the analyst even more dependent upon computers for effective control of the simulation and for interpretation of results.

The rapid growth of analytical methods and the exploitation of computer technology have paralleled an even faster expansion in complexity of engineering systems. Aerospace systems, moon missions, space satellite laboratories, nuclear reactor power systems, and automated chemical plants are examples of engineering systems that are so expensive and/or potentially hazardous that the design cannot be undertaken without computer simulation to predict ultimate performance. It is no longer feasible to restrict analysis to linearized or isolated subsystems in development of such systems. As technology continues to expand reliable methods must be found to predict, by simulation, the performance of total systems. For only with such analysis and prediction can decisions, involving capital investment as well as design features, be made. It is in this context that hybrid computation can be seen to fulfill a growing need.

One might well ask what are the implications of this escalation of complexity. If simulation models must necessarily grow larger, just how does this affect the procedures of analysis, computer programming, computation, and interpretation of results? How are the hazards, which were of earlier concern, of becoming overwhelmed with useless data and meaningless computation be avoided? There is a pernicious theory about programming for very large digital simulation, that says if two men can do the job in six months, four men will take twelve, and eight men would never complete it. How can the step from mathematical model to the first computer simulation run be held within bounds—to avoid inordinate investment in programming that may never work or may have to be scrapped for a better approach? How can the analyst or design engineer stay in touch with his model?

Surely there are no answers that yield guaranteed results. But these are serious questions and some direction is needed in order to evaluate properly the true potential of advanced computer techniques. The implications in the field of hybrid simulation may be divided into three categories:

(a) Model building in programming
(b) Software
(c) Automation

Model Building in Programming

The analyst, design engineer, and programmer of a large hybrid simulation must all (if they are more than one person) become involved in all phases of the simulation process. Responsibility cannot be divided up, as it often is at the digital "closed-shop" facility, between analyst and computer programmer. The hybrid computer laboratory must be operated, as many analog laboratories, on an open-shop basis with expert programming support available from the laboratory. The design engineer must have a genuine understanding of the computers to be used, even though he may not do the actual computer programming. Since the computer actually *becomes* the model of his system, he must know the limitations imposed by the machine as well as by the mathematics, and he must be able to communicate effectively with the computer. Moreover, during the construction of the mathematical model the analyst must keep in mind the features of the parallel and sequential parts of the hybrid computer in order to achieve a proper partitioning of the mathematical model to suit the computer.

Much attention has been given here to the relative speeds of computation inherent in the different computing techniques. It may be evident at this point that the presence of a very wide range of signal frequencies in a system to be simulated is the one characteristic that most clearly indicates the need for a hybrid computer. As an example, consider the simulation of a long range flight of a space capsule. In "real time" the position coordinates probably vary at 0.01 cps over most of the range, and, at most, at 3 cps during launch and re-entry. At the same time, pitch, roll, and yaw rates and thrust forces may reach 10 cps or more. Adaptive control functions and control surface forces may have transient frequencies as high as 50 cps; and a simulation of reaction jet control forces may require torque pulses as narrow as one millisecond. Since there is little or no damping in an orbital flight,these pulses have a long term effect, and accuracy in their representation is important. If an on-board predictive computer is included in the simulation, iterative calculations on the analog computer may involve signal frequencies of 100 to 1000 cps. Thus, this simulation spans a frequency range of 10^5 as well as a dynamic range in some dependent variables of 10^5 or 10^6 (Figures 14, 15, and 16).

The following observations may be fairly apparent, but in considering division of a problem between computers it is well to note the types of mathematics for which each is best suited. The forté of the digital computer is the solution of algebraic equations. If the equations are explicit, the calculation time is easily determined. Implicit equations often require a variable length of time, and if there are not too many of them they may be readily solved continuously on the analog computer. Numerical integration comes as a by-product of the computer's power in solving algebraic problems. Time is the only penalty. If the high precision is not needed, the integration is better done by the analog computer, for the solution of ordinary differential equations is its strong feature.

Evaluation of arbitrary functions is performed with ease by both computers, as well as by parallel digital components; however, if

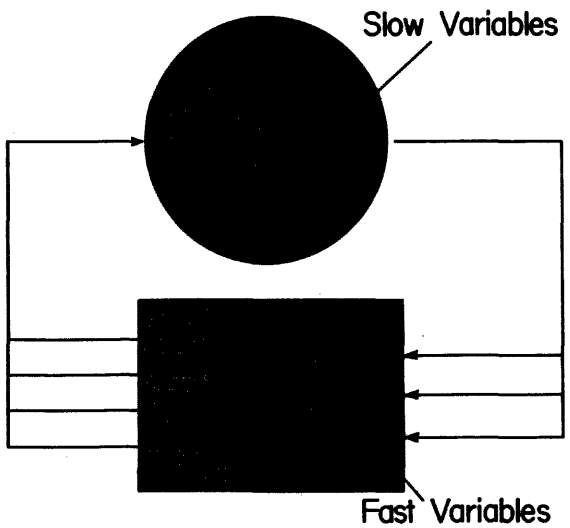


Figure 15. The relative speeds of computation inherent in the different computing techniques are important considerations in the planning of a computer simulation of a large complex system. The presence of a wide range of signal frequencies in a system to be simulated is the one characteristic that most clearly indicates the need for a hybrid computer.

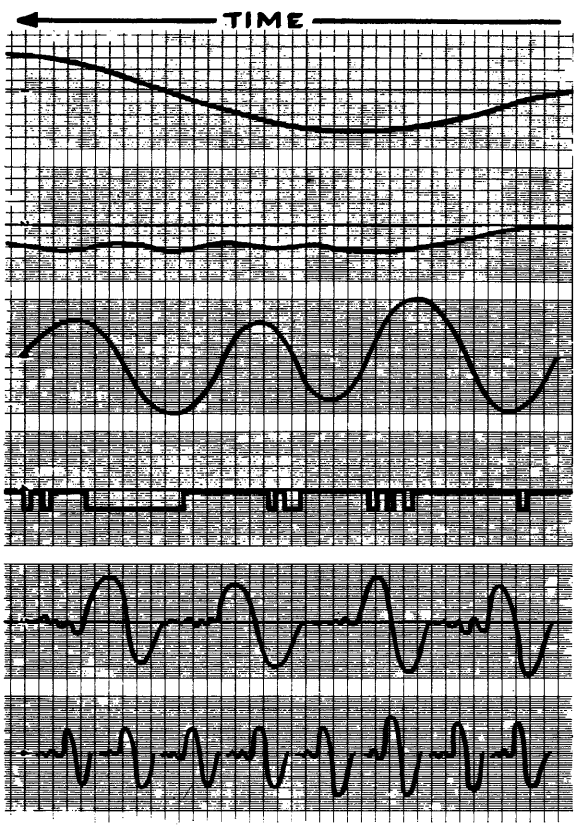


Figure 16. (Note: place figure horizontal with the word "time" on the left). A wide range of signal frequencies is suggested here: although not to scale, for a dynamic range of 10^6 could not be illustrated conveniently. Consider the simulation of a long range flight of a space capsule. The curves might represent (l. to r.): position coordinates varying at 0.01 cps; deviation from a desired path, 0.1 cps; pitch, roll, or yaw rate, 1-5 cps; reaction jet control pulses, 1 ms. pulses at several hundred pulses per sec.; thrust forces or control surface transients, 1-50 cps; iterative calculations for trajectory predictions, 100-1000 cps.

speed is important, and if the data are functions of two or more variables, a digital program is the best choice. Simulation of non-analytic non-linear functions, such as limits, back-lash, dead-zone, stiction, and hysteresis again are amenable to both techniques but analog is probably more economical. Evaluation of trigonometric and hyperbolic functions also can be done both ways and the choice seems to depend on the particular problem. In this case there is a third choice, for there are techniques and equipments for executing these functions by parallel digital components.[37]

Logic equations that must be evaluated continually with respect to their relation to analog variables clearly must be programmed with parallel logic elements. On the other hand, decision and control functions that have to do with the occasional evaluation of states of the computer and errors signals and with sequencing various sections of the total system through different modes and states of operation, may require both parallel and sequential operations.

Hybrid simulation requiring solution of partial differential equations opens a whole new subject for discussion. Let it just be said that although there is very little practical experience in this field, it appears to offer one of the most promising areas of growth for hybrid simulation. The digital computer approach to the solution of PDE is often limited by available computer time—particularly in simulation problems where it is desired to solve the problem many times for various conditions. The analog computer can solve some PDE problems very efficiently but it is often seriously limited by the necessity of large amounts of equipment for complex problems. Moreover, only with memory to store complete functions (either in a parallel digital system or a sequential computer) can certain kinds of boundary value problems be approached. The hybrid computer has the ability to store boundary conditions as well as complete sets of intermediate solutions so that parallel computer programs can be used for speed, but then be time-shared over again for different parts of the space domain. Some very challenging problems and some promising possibilities face the experimenter in this field.

Returning now to the implications of the growth of complexity in simulation, the important point in programming is that whoever prepares the computer program must himself be a model builder and be familiar with the system to be simulated. The program should be designed, constructed, and checked out in much the same way that any other kind of model of a complex system would be built. The computer model should be put together from working models of subsystems. Each subsystem, or group thereof, is verified for correct performance in a linear or simplified mode before connecting it to other parts of the model. At each point in the expansion of the model,

including the final one, at least one test case of a linear mode of operation should be checked against known or precalculated behavior.

Computers can extend the analytical powers of man, but they cannot work magic. The computer is a storehouse of answers, but the burden is upon the analyst to ask meaningful questions if he is to receive useful answers. Only with a step by step process, asking questions of the computer at each step, can a valid and useful computer model be built for a large, complex system.

Software

Standard programs and routines for digital computers, of general utility to programmers, known as "software," are in such wide use that the production of software is virtually an industry in its own right. "Automatic programming systems," which make computer programming easy for the non-computer expert, are responsible for the almost universal acceptance of digital computers in scientific research and development.

The development of sophisticated software has made it possible to increase computer utilization, to gain wider use of computers with minimum training of personnel, and to reduce duplication of programming effort for programs of general utility. A total dependence upon automatic programming has the disadvantage of isolating the problem analyst, and even the programmer himself, from the computer. The analyst is restricted from communicating with his computer model while computation takes place. The programmer is limited in taking full advantage of the computer's special features.

On the other hand, in the analog computer field the reverse situation exists. No comparable "software" usage has shown up. There is little in the way of automatic programming and preserving of standard programs by analog computer programmers. However, another characteristic of simulation by analog computer is that the problem anayst is involved in building the computer model and he maintains rapport with his model during the simulation.

The role that software must play in hybrid simulation of large complex systems is evident. "Hybrid software" must ease the programmer's burden, as it does for the digital programmer, and at the same time it must bring the analyst closer to his model rather than isolating him from it. Hybrid software must include not only coding for the sequential computer but also interconnection diagrams and prewired patch panels for the parallel machine. The following types of software are needed to support growth of hybrid computation to meet the simulation needs of today.

a. *Compilers and Assemblers.* Conventional compilers and intepreters have their useful place in hybrid computation, to aid in processing data prior to computation and processing results for interpretation. Automatic programming systems for hybrid computation may differ from conventional systems only in three ways: running time of the object program is minimized at the expense of compiling time; actual running time for each program statement is precalculated or estimated to aid the programmer with timing of the parallel/sequential interface; and while the programmer's language is "problem oriented," it is required to be machine-dependent. The programmer must be able to utilize special machine features and to program for control of all interface operations.

b. *Utility Library.* In addition to the conventional utility routines for mathematical functions, format conversions, and input/output operations, the hybrid simulation library should expand with routines for specific transfer functions of useful subsystem models that have become standard and are used in larger models, e.g., a typical servo controller. Another type of example is a function generator program to any number of aerodynamic functions. A standard program for a complete aerodynamic vehicle simulation is also feasible.

c. *Input/Output Routines.* Direct on-line control of the computer model by the analyst is needed. In a convenient language it must be possible to experiment with time scales, parameter values, and even to make substitution of mathematical algorithms (particularly integration algorithms), without any penalty in running time. This means that a complete symbol table and all definitions of parameters must be available to an executive routine that will accept operator instructions to modify a particular *problem variable* and proceed to calculate

changes in all the *"machine variables"* that are affected. The executor also permits interrogation of the state (and time history) of *any problem variable,* in engineering units. Upon operator command the model can be modified by changing the linkages between submodels or subroutines. It may, for instance, be desired to linearize part or all of the model for checking purposes, or to isolate or "freeze" certain parts of the model. In monitoring and adjusting the model the executor must not be limited to the sequential computer but should have full access to the parallel elements in the computer system.

Automation

One important aspect of hybrid computation, which is barely mentioned above, is the opportunity for automating much of the routine parts of programming and check-out of the analog computer. It is evident that a necessary feature of any hybrid computer system is the mechanization of, and sequential program control of, as many of the manual operations on the analog computer as possible. This includes setting of potentiometers, switches, modes, time scales, recorders, and the selector and readout system. This kind of control is important for some of the software functions mentioned above. Also it makes possible automatic set-up, testing, and diagnosis of machine and program faults. Some interesting diagnostic programming for such a hybrid computer was developed in 1959-60 at General Electric MSVD in Philadelphia.[15,17,32]

A different type of programming automation is offered by the Apache system developed at Euratom, Ispra, Italy, for the IBM 7090 and PACE analog computers.[14] This is a digital program that translates a mathematical statement into detailed programming instructions for the analog computer. While Apache is not intended for hybrid computing the appropriateness of such a program should be apparent.

One last important characteristic of hybrid simulation concerns the automation of the model building process itself. Simulation inherently involves trial and error experimentation. The elements of a model are verified; sensitivity to environment is explored; and variation of performance due to parameter changes are evaluated. When a criterion for optimality can be specified, experiments are made to obtain optimum performance. The sequential computer is perfectly suited to the automation of these procedures. Between calculations the digital computer can evaluate the results, decide upon changes to the model or the data, and implement the changes. At the same time the analyst can monitor the progress of the simulation and interrupt the automatic process whenever human judgment is required.

CONCLUSIONS

The main points developed in this paper may be listed simply.

a. Hybrid computation is built upon the technology of analog and of digital computers and is equally dependent upon the programming methods, software, and procedures of problem analysis that have been developed for each.

b. A hybrid computer is a compatible system of *parallel* computing components, both digital and analog, and a stored program *sequential* machine. The hybrid programmer must be constantly aware of the relative timing of computational events in the parallel and sequential parts of the system.

c. There is an ever growing need for simulation of very complex engineering systems. The process of analysis and building of a computer model for evaluation and prediction of behavior are a required step in many large development programs. The hybrid computer offers a means for many such simulations that would be impractical by other means. Hybrid computation is inherently a tool for very complex simulations rather than simple studies.

BIBLIOGRAPHY

1. BARNETT, R. M., "NASA Ames Hybrid Computer Facilities and Their Application to Problems in Aeronautics," *International Symposium on Analogue and Digital Technique Applied to Aeronautics,* Liege, Belgium (September 1963).
2. BAUER, W. F., and WEST, G. P., "A System for General Purpose Analog-Digital Computation," JACM, Vol. 4, No. 1 (January 1957), p. 12.

3. BAXTER, D. C., and MILSUM, J. H., "Requirements for a Hybrid Analog-Digital Computer," National Research Council of Canada, Mechanical Engineering Report MK-7, Ottawa (October 1959), and American Society of Mechanical Engineers, Paper No. 59-A-304.
4. BERTRAM, J. E., "Effect of Quantization in Sampled-Feedback Systems," *U.S. Government Research Reports,* Vol. 31 (January 16, 1959), p. 19 (A), PB 133 341.
5. BIRKEL, G., JR., "Hybrid Computers for Process Control," *Communication and Electronics,* No. 52 (January 1961), pp. 726-734; discussion, p. 734.
6. BLANYER, C. G., and MORI, H., "Analog, Digital, and Combined Analog-Digital Computers for Real-Time Simulation," Proc. EJCC (December 1957), p. 104.
7. BURNS, A. J., and KOPP, R. E., "A Communication Link Between an Analog and a Digital Computer (DATA-LINK)," Grumman Aircraft Engineering Corp., Research Department, Research Report RE–142 (October 1960), *ASTIA,* No. AD 244 913.
8. _____, "Combined Analog-Digital Simulation," Proc. EJCC, Vol. 20 (December 1961), pp. 114-123.
9. CAMERON, W. D., "Determination of Probability Distribution Using Hybrid Computer Techniques," *International Symposium on Analogue and Digital Techniques Applied to Aeronautics,* Liege, Belgium (September 1963).
10. CONNELLY, M. E., "Simulation of Aircraft," Servomechanisms Lab. Report 7591-R-1, M.I.T. (February 15, 1959).
11. _____, "Analog-Digital Computers for Real Time Simulation," M.I.T. Report DSR8215, Final Report, ESL-FR-110 (June 1961).
12. _____, "Real-Time Analog-Digital Computation, IRETEC, Col. EC-11, No. 1 (February 1962), p. 31.
13. COX, F. B., and LEE, R. C., "A High-Speed Analog-Digital Computer for Simulation," IRETEC, Vol. EC-8, No. 2 (June 1959), pp. 186-196.
14. DEBROUX, A., and GREEN, C., and D'HOOP, H., "Apache—A Breakthrough in Analog Computing," IRETEC, Vol. EC-11, No. 5 (October 1962).
15. FEUCHT, K., "Diagnostic Programs for a Combined Analog-Digital System," *Proceedings of the Combined Analog Digital Computer Systems Symposium,* Philadelphia (December 16-17, 1960).
16. GAINES, W. M., and FISCHER, P. P., "Terminology for Functional Characteristics of Analog-to-Digital Converters," Control Engineering, Vol. 8, No. 2 (February 1961), pp. 97, 8.
17. GELMAN, H. D., "Evaluation of an Intercept Trajectory Problem Solved on a Combined Analog-Digital System," *Proceedings of the Combined Analog Digital Computer Systems Symposium,* Philadelphia (December 16-17, 1960).
18. GELMAN, R., "Corrected Inputs—A Method for Improving Hybrid Simulation," Proc. FJCC, Vol. 24 (November 1963).
19. GREENSTEIN, J. L., "Application of ADDA Verter System in Combined Analog-Digital Computer Operation," Pacific General Meeting, AIEE (June 1956).
20. HALBERT, P. W., "Hybrid Simulation of an Aircraft Adaptive Control System," Proc. FJCC, Vol. 24 (1963).
21. HARTSFIELD, E., "Timing Considerations in a Combined Simulation System Employing a Serial Digital Computer," *Proceedings of the Combined Analog Digital Computer Systems Symposium,* Philadelphia (December 16-17, 1960).
22. KORN, G. A., "The Impact of the Hybrid Analog-Digital Techniques on the Analog Computer Art," Proc. IRE, Vol. 50, No. 5 (May 1962), pp. 1077-1086.
23. LANDAUER, J. P., "Simulation of Space Vehicle with Reaction Jet Control System" (1962), EAI Bulletin No. ALHC 62515.
24. _____, "The Simulation of Transport Relay with the Hydac Computing System," EAI Bulletin No. ALHC 63011.
25. MCLOED, J. H., and LEGER, R. M., "Combined Analog and Digital Systems—Why, When, and How," *Instrument and Automation,* Vol. 30 (June 1957), pp. 1126-1130.

26. MITCHELL, B. A., "A Hybrid Analog-Digital Parameter Optimizer for ASTRAC-II," Proc. SJCC, Vol. 25 (April 1964).

27. MITCHELL, J. M., and RUHMAN, S., "The Trice—A High Speed Incremented Computer," *IRE Nat. Conv. Record* (1958), Pt. 4, pp. 206-216.

28. MIURA, T., IWATA, J., "Effects of Digital Execution Time in a Hybrid Computer," Proc. FJCC, Vol. 24, (November 1963).

29. NORONHA, L., "An Integrated General Purpose Hybrid Computing System," *International Symposium on Analogue and Digital Techniques Applied to Aeronautics*, Liege, Belgium (September 1963).

30. PALEVSKY, M., "Hybrid Analog-Digital Computing Systems," *Instruments and Automation*, Vol. 30 (October 1957), pp 1877-1880.

31. _____, "The Digital Differential Analyzer," *Computer Handbook*, Edited by G. A. Korn and H. O. Huskey, New York: McGraw-Hill (1961), Chapt. 19.

32. PASKMAN, M., and HEID, J., "Combined Analog-Digital Computer System," *Proceedings of the Combined Analog Digital Computer Systems Symposium*, Philadelphia (December 16-17, 1960).

33. SHAPIRO, S., and LAPIDUS, L., "A Combined Analog-Digital Computer for Simulation of Chemical Processes," *Proceedings of the Combined Analog Digital Computer Systems Symposium*, Philadelphia (December 16-17, 1960).

34. SHILEIKO, A. V., "A Method for Selecting the Optimum Structure of a Digital Analog Computer," *Automation and Remote Control (Avtomatika i Telemekhanika)*, Vol. 22, No. 1 (August 1961), (originally published January 1961), pp. 76-81.

35. SKRAMSTAD, H. K., ERNST, A. A., and NIGRO, J. P., "An Analog-Digital Simulator for Design and Improvement of Man-Machine Systems," Proc. EJCC (December 1957), p. 90.

36. SUSSKIND, A. K., "Notes on Analog-Digital Conversion Techniques," M.I.T., Technology Press (1957).

37. VOLDER, J. E., "The Cordic Trigonometric Computing Technique," IRETEC (September 1959), pp. 330-334.

38. WEST, G. P., "Computer Control Experience Gained from Operation of a Large Combined Analog-Digital Computation System," *Proc. of Computers in Control Systems Conference*, Atlantic City (October 1957), p. 95.

39. WILSON, A., "Use of Combined Analog-Digital System for Re-entry Vehicle Flight Simulation," Proc. EJCC, Vol. 20 (December 1961), pp. 105-113.

40. WITSENHAUSEN, H., "Hybrid Simulation of a Tubular Reactor" (1962), EAI Bulletin, No. ALHC 6252-25.

41. _____, "Hybrid Techniques Applied to Optimization Problems," Proc. SJCC, Vol. 21 (May 1962).

42. WORTZMAN, D., "Use of a Digital/Analog Arithmetic Unit within a Digital Computer," Proc. EJCC (December 1960), p. 269.

43. ZETKOW, G., and FLEISIG, R. (Grummen Aircraft Corp.), "Dynamic Analysis of OAO Spacecraft Motion by Analog-Digital Simulation," IRE Convention, Space Electronic Session (March 1962).

A HYBRID ANALOG-DIGITAL PARAMETER OPTIMIZER FOR ASTRAC II

Baker A. Mitchell, Jr.
Department of Electrical Engineering
University of Arizona
Tucson, Arizona

INTRODUCTION

This paper describes an optimizer designed to find system parameter combinations which optimize a functional, F, such as

$$F(a_1, \ldots, a_n) = \int_0^T [y^2(t) + u^2(t)]\, dt$$

where

$$y(t) = y(t, a_1, \ldots, a_n)$$
$$u(t) = u(t, a_1, \ldots, a_n)$$

are state and control variables depending on the unknown parameters $a_1, \ldots, a_n$ in accordance with the system equations

$$\dot{y}_i = f_i(y_1, \ldots, y_k; u_1, \ldots u_m; t)$$

The new optimizer is designed to work with a fast all-solid-state iterative differential analyzer (ASTRAC II) which is capable of producing complete solutions $y_i(t)$ and the corresponding values of the performance measure $F(a_1, \ldots, a_n)$ for up to 1000 new parameter combinations per second.[1]

To simplify optimizer logic and memory requirements in problems involving many parameters, we simultaneously implement random perturbations[2] on all parameters a_k and step to the perturbed point whenever the perturbation yields an improvement in the performance measure $F(a_1, \ldots, a_n)$.

Simple all-digital logic permits implementation of different sequential optimization strategies, including correlation between random-perturbation vectors and step-size changes depending upon past successes and failures. The analog integrator/multipliers commonly used to set system parameters have been replaced by simple, reversible binary counters driving D/A converters[3] for simplified design and improved reliability.[4] The principle of the optimizer is shown in the block diagram of Fig. 1.

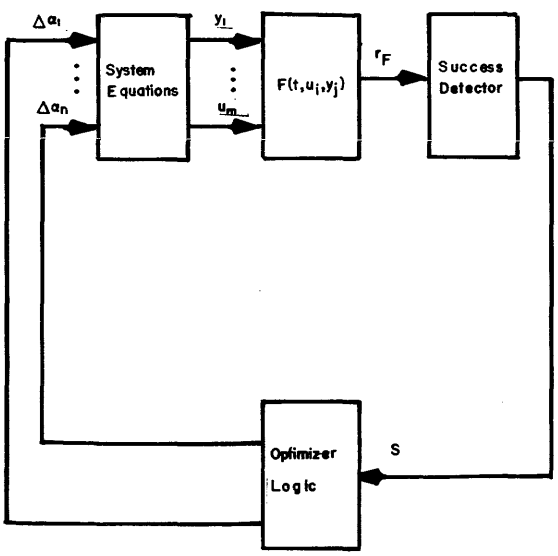


Figure 1. Optimizer Block Diagram.

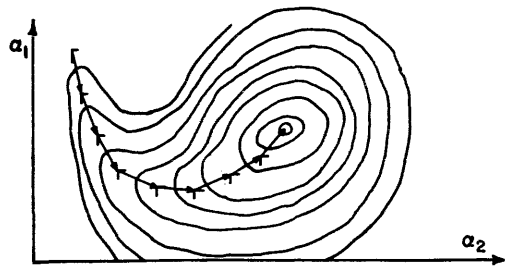


Figure 2a. Optimum Gradient Method.

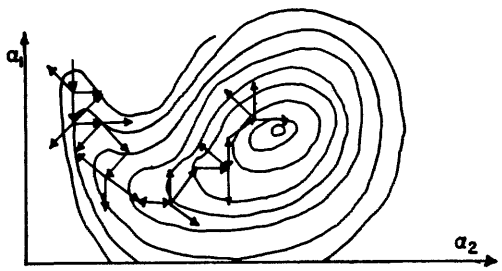


Figure 2b. Pure Random Perturbations.

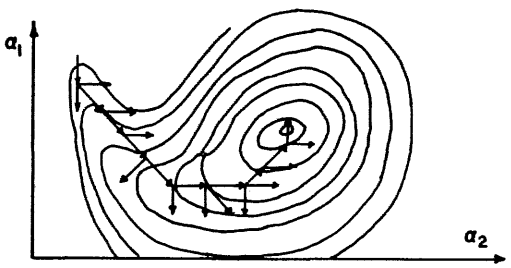


Figure 2c. Correlated Random Perturbations.

Figure 2. Typical Optimization Paths.

Figure 2a shows the parameter-space path over which a conventional deterministic system would optimize a simple two parameter system. Starting with a trial set of parameters ${}^{0}a_i$ the conventional optimization logic employs the results of successive differential analyzer runs to obtain succeeding parameter values

$$ {}^{r}a_1 = {}^{r-1} + {}^{r}\triangle a_1 $$

which successively improve ${}^{r}F = F({}^{r}a_1, \ldots, {}^{r}a_n)$.

The most frequently used method employs n *trial steps* to compute approximate gradient components $\triangle F/\triangle a$ in each parameter direction; these gradient components are then stored and used to compute the optimal correction $\triangle a_i$ for a *working step,* or for a series of working steps in the same direction.[8]

Such deterministic methods require complex logic and storage. Although they may converge well for favorable performance functions $F(a_1, \ldots, a_n)$, they may "hang up" on ridges or in canyons of the multidimensional landscape of the performance measure domain.[5] Furthermore, if the performance measure contains discontinuities, nonlinearities, or large higher-order derivatives with respect to the parameters, our information of past performances will be of little value in determining succeeding steps; thus the step-size may have to be reduced to such a degree that convergence to the optimum is extremely time-consuming.

Figure 2b shows how a pure random-perturbation scheme might optimize the same function. Here, $\triangle a_i$ may be positive, zero, or negative with equal probability; and the nominal parameter point is moved as soon as the first improvement in the performance function occurs. No attempt is made to affect perturbations by past results or gradient methods.

On the other hand, if perturbations are to be correlated with past successes or failures, then the optimization path might appear as shown in Fig. 2c. Such a scheme causes future increments $\triangle a_i$ to favor the direction in which past improvements in the performance function were made. Notice, however, that we still do not require computation of individual gradient components, as in deterministic gradient optimization schemes. Hence, logic and memory requirements are reduced.

Motivation for Random Search

The basis for all direct computer methods of parameter optimization is the same: using a mathematical model or simulated system, we set the parameters to some trial values and compute the performance criterion. Then according to some *rule,* we reset the parameters to new values and again compute the performance criterion. This procedure is repeated until some desired degree of improvement is obtained.

Naturally it is hoped that the rule for adjusting the parameters will take *maximum* advantage of the knowledge gained from observing previous trials, and by so doing achieve the optimum set of values for the parameters in the shortest possible time. Usually, however, this rule depends *solely* upon the knowledge gained from recent past trials and this is thought to be equivalent to using this knowledge to maximum advantage. If, however, the performance criterion contains discontinuities, nonlinearities, or large higher-order partial derivatives with respect to the parameters, our information of recent past behavior (actually a total or partial *first* derivative) may be of little value for the determination of successive steps; and if the step size is too large, this information from the preceding step (or from a short forward trial step) will be totally misleading. Thus, with conventional, deterministic perturbation such as the gradient method, one may be forced to reduce the step size to such a degree that convergence to the absolute optimum is excessively time-consuming. For this reason it has been suggested that randomness be introduced into the search rule—perhaps in proportion to the expected severity of the discontinuities, nonlinearities, etc., present in the cost function domain.

In reality, it may be difficult to make any reliable prediction concerning the behavior of the performance function. Even with reasonably well-behaved performance functions, it can be quite difficult to foresee "ridges," "temporary plateaus," "saddle-points," and other features which render deterministic rules far from foolproof.

Reference 5 goes further into such motivation for random-search methods (Figs. 3 and 4).

PRINCIPLES OF OPERATION

Most of the optimization strategies proposed here can be based on the flow chart of Fig. 5. The operation common to all the various strategies consists of incrementing all parameters simultaneously by individual random increments $+\triangle a$, $-\triangle a$, or zero; the common magnitude $\triangle a$ (step size) of these increments is subject to a separate decision. The term *success* will be used to indicate that the incremented set, $a_i + \triangle a$, has yielded a more favorable value for the performance measure than was obtained with the unincremented set, a_i. A *failure* will mean that the incremented set yielded a less favorable value for the performance measure, in which case the failing increment is subtracted from the parameter before a new increment is added. Successive failures and successes can be counted and used to decide when an increase or decrease in the step size might be

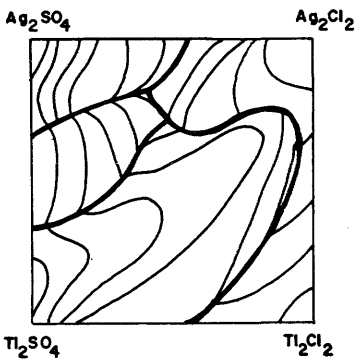


Figure 3. The function defined represents a system composed by particular percentages of each of the four compounds shown at the corners. Contour lines are drawn in order to indicate values of a property on the system. The heavy lines are lines of discontinuity in slope.[5]

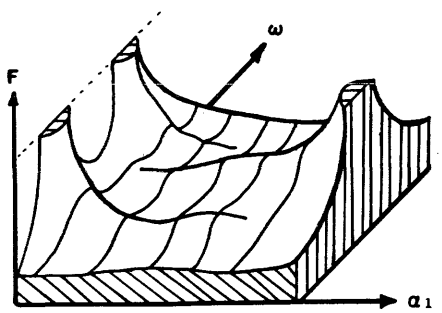


Figure 4. This sketch of a dynamic vibration absorber shows the amplitude of vibration F plotted over the frequency range ω for values of the parameter a_1. The effect of only one of the three parameters, a_1, a_2, a_3, of the actual system could be drawn. A possible criterion for an optimum absorber is to require that the maximum amplitude of vibration yielded by a particular set of parameter values, a_1, a_2, a_3, be minimum over the frequency range of interest.[5]

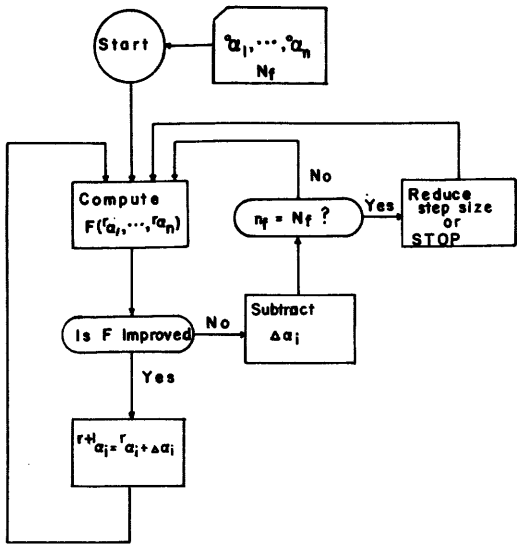


Figure 5. Flow Diagram for a Typical Optimization Routine.

advantageous. A binary noise generator[7] together with digital correlation logic decides what the sign of each new increment will be. Thus, the overall effect of the perturbation scheme is an n-dimensional random walk.

THE ASTRAC II SYSTEM

The dynamic system and the performance criterion are to be simulated on the Arizona Statistical Repetitive Computer, ASTRAC II, although ASTRAC I served for preliminary studies. ASTRAC II is a ±10 volt, all-solid-state iterative differential analyzer capable of iteration rates of 1 Kc. as well as real time computation. The first 20-amplifier section of ASTRAC II is to be completed in the fall of 1964.

The *analog section* has a large conventionally appearing patchbay. 20 Mc. transistorized amplifiers mounted in shielded cans plug directly into the rear of the patchbay without any intervening wiring. The analog section will comprise sample-hold memory pairs, comparators, analog switches, switched integrators, diode quarter-square multiplifiers, and diode function generators.

Timing and logical control is furnished by the *digital section,* which provides timing pulses, integrator RESET pulses, and sampling pulses. The digital section has its own patchbay with removable patchboards for implementing various logic functions. Patchable gates and flip-flops are used in conjunction with the prewired timing and RESET circuits. In view of the amount of logic involved in the optimizer, however, it was thought best to build it as a separate digital section with its own removable patchboards, devoted to this purpose. The complete ASTRAC II optimizer will not only implement the sequential random search optimization described here, but will permit comparison with deterministic optimization schemes.

Optimizer Logic

The digital logic of the optimizer is subdivided into basic functional units whose inputs, outputs, and control points are wired to its patchbay (Fig. 6). With a different prepatch panel, these components are also available for other uses besides optimization. In particular, the parameter-setting circuits will also serve for experiments with deterministic optimization schemes.

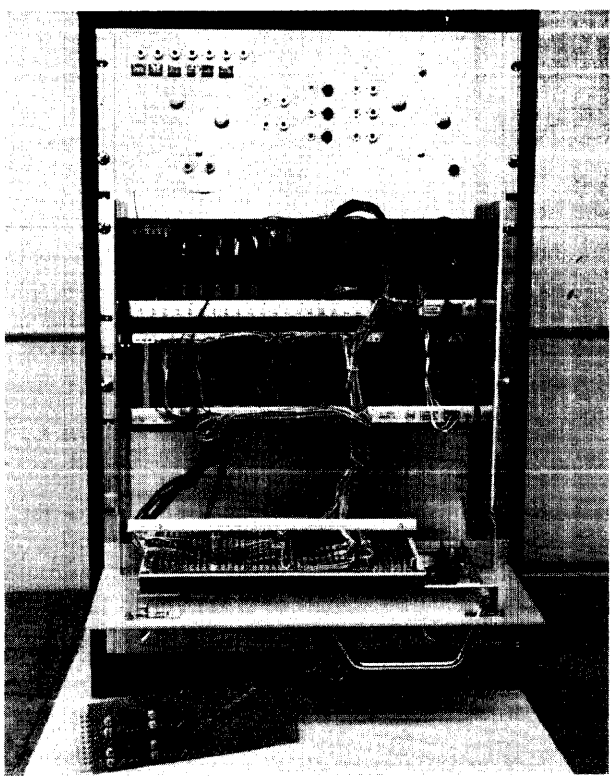


Figure 6. Photograph of the Optimizer. A D/A multiplier card is shown at bottom left.

The functional units are built of commercial plug-in logic cards interconnected on racks with wire-wrap terminations for ease of modification and expansion.

The resistor networks and switches comprising the D/A multipliers are mounted in shielded plug-in cans adjacent to the operational amplifiers behind the analog patchbay. Shielded digital control lines connect each D/A multiplier to the optimizer patchbay.

Hybrid Analog-Digital Noise Generator

ASTRAC II will employ a new noise generator[10] producing pseudo-random maximum-length shift-register sequences at any desired clock rate up to 4 Mc. We may obtain either a single pseudo-random sequence repeating after 33 million bits, or four uncorrelated sequences one-fourth as long.

Success-Failure Indicator

Essentially, the function of this circuit (Fig. 7) is to compare the value of the best performance measure, ${}^{L}F$, obtained to date, with the performance measure just yielded by the last computer run, ${}^{r}F$. If the last run was a *failure,* (e.g., ${}^{r}F < {}^{L}F$ for maximization), then U_3 and $\tilde{S}$ remain zero; and ${}^{L}F$ is still held as being the best value. If, however, the last computer run was a *success* (${}^{r}F > {}^{L}F$), then U_3 becomes "1" which causes the second sample-hold to take on the value just obtained, ${}^{r}F$, as now being the new optimum. The pulse U_3 also causes flip-flop 7 to assume the "1" state (s-o) which indicates to the digital optimization logic that a success was obtained with the last set of incremented parameters.

If it is known that F will always be monotonically increasing during the latter part of the COMPUTE period, Switch 3 and U_2 need not be used.

Master Clock

The Master Clock provides all timing pulses throughout the digital optimization routine. It consists of a four bit Gray-code counter driven by a 1 MC. pulse, C_1, from the differential-analyzer digital control unit. C_1 is gated to the counter during the differential-analyzer RESET period. With the exception of the Success-Failure Indicator circuit, where timing is under

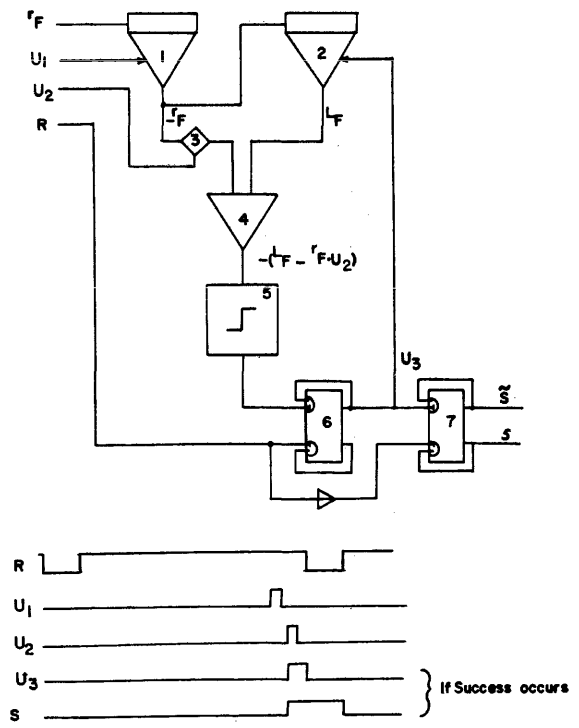


Figure 7. Success-Failure Indicator.

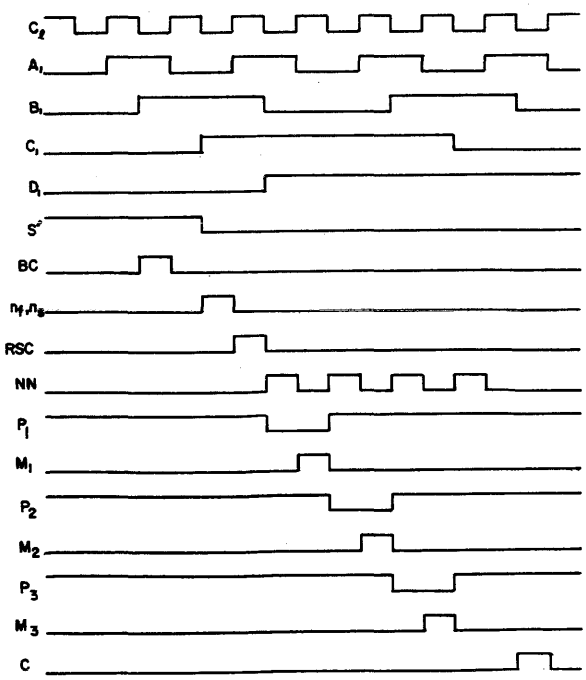


Figure 8. Pulse Sequence from Master Clock.

control of the differential-analyzer, no optimization *logic* is performed during the COMPUTE period of the differential analyzer.

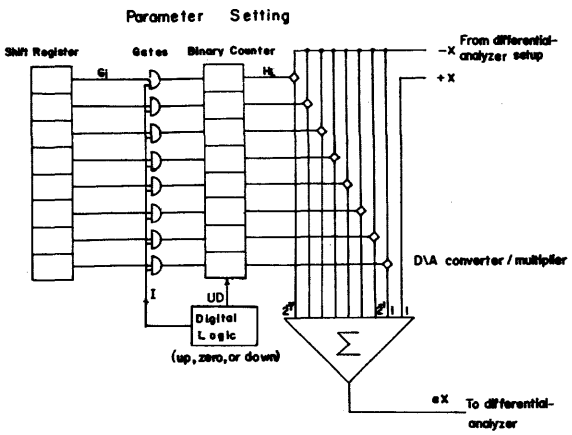


Figure 9. Parameter Setting.

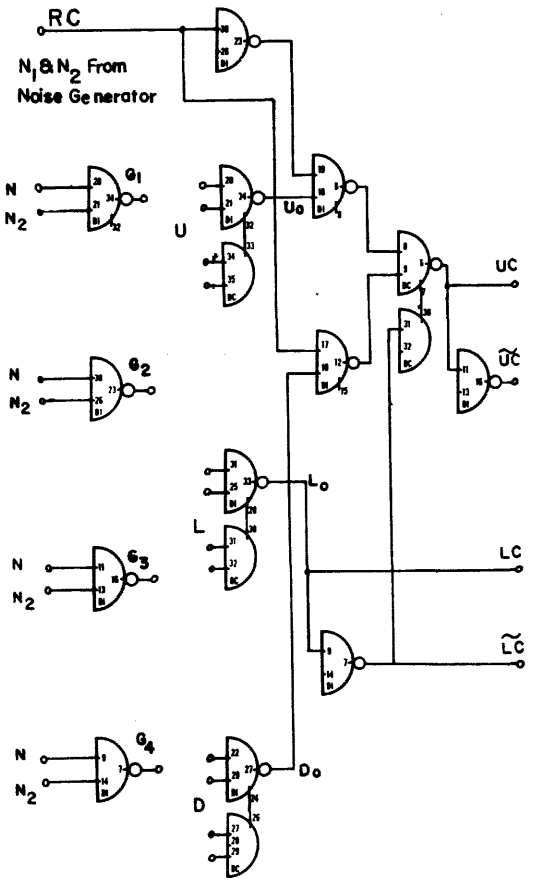


Figure 10a. U-L-D Selector Circuit.

The sequence of the timing pulses from the Master Clock is shown in Fig. 8. S′, as shown, occurs only after failures, i.e., S = 1; if S = 0, S′ = 1 throughout the RESET period. A success or failure also causes n_S or n_F to occur (Fig. 15).

PARAMETER SETTING

The flexibility needed to implement a variety of different optimization-logic schemes while maintaining simplicity, reliability, and low cost is achieved by using a unique method of parameter setting.

Binary-Counter Operation

Referring to Fig. 9, the binary up-down counter increments whenever a pulse appears on the "I" line. The right-left shift register contains zeros in all except one of its stages, and the position of this "1" selects the stage of the binary counter which is to receive the "I" pulse. By controlling the D/A multipliers, the counter has then increased or decreased α_i by an

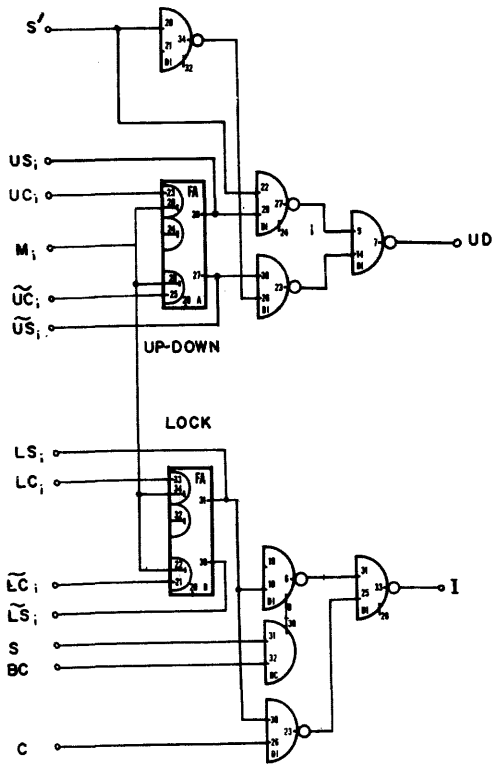


Figure 10b. U-L-D Memory.

increment $\triangle a_i$; the magnitude $\triangle$ is determined by the position of the "1" in the shift register, and the sign (+ or —) is determined by the logic level on the UD line (1 = count up, 0 = count down).

U-L-D Digital Logic

The U-L-D (UP-LOCK-DOWN) Logic is composed of two sections: a central *U-L-D Selector Circuit* (Fig. 10a), and a *U-L-D Memory* associated with the binary counter for each parameter (Fig. 10b).

The selector circuit accepts two uncorrelated random bits $^1N_1$, $^1N_2$ obtained from the ASTRAC II noise generator. Depending on the interconnections of the selector-circuit gates G_1, G_2, G_3, G_4 to gates U, L, D, the gate outputs U_0, L_0, D_0 will have different joint probability distributions as shown in Table I.

TABLE I

P(U = 0)	P(L = 0)	P(D = 0)	Connect	to	Gates
1	0	0	G_1, G_2, G_3, G_4		U
¾	¼	0	G_1, G_2, G_3 G_4		U L
¾	0	¼	G_1, G_2, G_3 G_4		U D
½	½	0	G_1, G_2 G_3, G_4		U L
¼	¾	0	G_1 G_2, G_3, G_4		U L
½	¼	¼	G_1, G_2 G_3 G_4		U L D
¼	½	¼	G_1 G_2, G_3 G_4		U L D
½	0	½	G_1, G_2 G_3, G_4		U D
0	1	0	G_1, G_2, G_3, G_4		L
¼	¼	½			
0	¾	¼			
0	½	½			
¼	0	¾			
0	¼	¾			
0	0	1			

Effectively, these distributions can be obtained by placing a logical "1" on the RC line.

The gate outputs, U_0, L_0, D_0, determine the states of the dc set and reset level controls on the UP-DOWN flip-flops and the LOCK flip-flops of all U-L-D Memories (one for each parameter) simultaneously. The pulse M_1 from the Master Clock now first sets the U-L-D flip-flops of the *first* parameter to their proper states as determined by the first set of random bits ${}^1N_1$, ${}^1N_2$.

Next, the noise generators are pulsed again by the Master Clock (pulses NN) producing two new random bits, ${}^2N_1$, ${}^2N_2$, which determine a new set of states for the dc set and reset level controls of the U-L-D Memory flip-flops. Now, the pulse M_2 to the U-L-D Memory of the *second* parameter sets *its* UP-DOWN and LOCK flip-flops in accordance with ${}^2N_1$, ${}^2N_2$. The process repeats for the remaining parameters.

U-L-D Memory

Note that we shall require two decisions for each parameter. The LOCK circuitry decides whether a_k is to be incremented or not incremented ("locked"). The UP-DOWN circuits decide the direction (up or down) of the increments $\triangle a_k$, if any. We shall consider the UP-DOWN circuits first.

If an UP-DOWN flip-flop is set (reset), a logical 1 (0) will appear on the "UD" line to the binary counter, *if* $S' = 1$. We now come to the LOCK decision. If the LOCK flip-flop is set or reset the incrementing commands "C" and "BC" from the Master Clock will or will not carry through the gates on "I" to increment the counters in the direction dictated by "UD"

Correlation Circuit

This circuit can be patched so as to introduce correlation between successive random perturbations according to some strategy selected to speed optimization. Suppose that the last set of increments succeeded in improving the performance function. Assuming that the performance function is fairly well-behaved, the greatest chance for another success lies in weighting each parameter so that it will probably increment in the same direction as in the previous trial (strong positive correlation). By the same reasoning, after a failure a strong negative correlation might be introduced. After either a failure or a success, if a particular parameter had been *locked* (i.e., its last increment was zero) then a probability function giving equal weight to all three states might be desirable for the next trial.

Referring to Fig. 11 the correlation circuit is patched as desired and senses the sign of the last increment in each U-L-D Memory, starting with the first parameter. If $\triangle a_k$ is positive, the RC line changes the selector gating so as to increase the chance of a positive $\triangle a_k$ for the next run if the last run was successful. The reverse can take place if the last run was a failure, depending upon the correlation control "CC".

If, however, $\triangle a_k$ was zero, as indicated by the state of the LOCK flip-flop, then the correlation line RC is turned on or off with equal probability by a random-noise input. This entire process is repeated for each parameter in turn, as they are sequenced by the pulses P_i from the Master Clock.

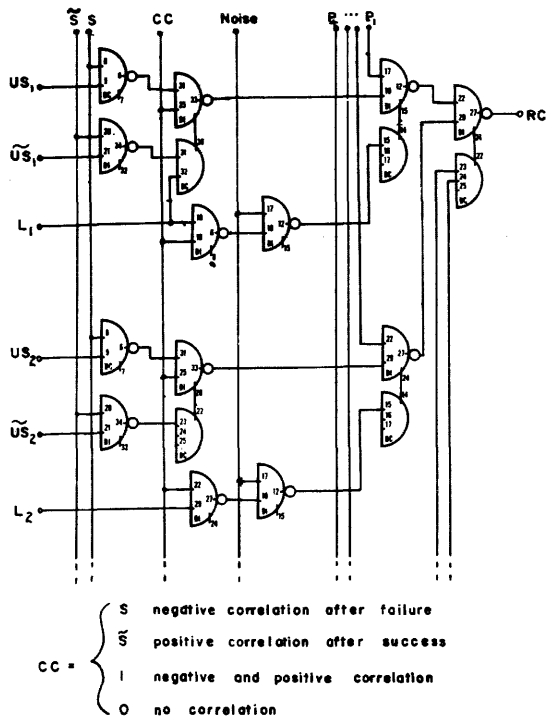


Figure 11. Correlation Circuit.

OPTIMIZATION STRATEGIES

1. Pure Random Perturbations

Pure random perturbations are achieved if we preset the "1" in the shift register to gate the count pulse into the binary counter stage yielding the desired step distance, e.g., ½, ¼, ..., 1/128. The noise generator is connected *directly* to the SET and RESET level controls of the Up-Down flip-flop of each parameter. During the analog RESET period, the noise generator is pulsed once for each parameter shortly before the sequenced pulse to the proper Up-Down flip-flop arrives. Thus, the Up-Down flip-flops sequentially assume the successive states of the noise source. The count pulse, C, is then fed to all parameters simultaneously. Hence, all parameters increment by the same magnitude but with random signs during each analog RESET period, executing a random walk in the perturbation scheme.

With pulses S, S′ and BC added to the U-L-D Memory, the set of increments used in the preceding run can be subtracted from the counters prior to the addition of increments for the next run. Thus, after a failure in an optimization run, the counters can be returned to the state which yielded the last success, before making another search.

2. Random Walk with Reflecting Barriers

This scheme is exactly the same as the pure random walk with one additional operation. Prior to assigning a new set of signs to the Up-Down flip-flops, a pulse to signify that a parameter has reached a barrier—possibly from a comparator in the analog system—can be gated to the U-L-D logic of the parameter to be reflected, causing its Up-Down flip-flop to complement and, later, to ignore the sign that is assigned to it by the noise source. In this manner, the parameter will have been *reflected* to its old position held two analog COMPUTE periods previously.

3. Random Walk with Varied Step Size

This scheme also contains only one addition to the pure random walk. When it is desired to increase or decrease the step size—possibly after a certain number of successive failures have been counted—one has only to insert the increase or decrease command, e.g., the counter output, into the shift-right or shift-left input of the shift register. This gates the count pulse either to the next higher or next lower parameter counter stage. The noise generator then assigns polarities to the Up-Down flip-flops, and the parameters are incremented by the new step size.

For this purpose, the optimizer contains two success-failure counters.

4. Correlated or Biased Random Walk

Since two independent noise sources are available, one can arrange $P(N_1N_2) = \frac{1}{4}$, $P(N_1\widetilde{N}_2) = \frac{1}{4}$, $P(\widetilde{N}_1N_2) = \frac{1}{4}$, $P(\widetilde{N}_1\widetilde{N}_2) = \frac{1}{4}$. These can combine to give $P(X) = \frac{1}{2}$, or $P(Y) = \frac{3}{4}$, where $X = N_1N_2 + N_1\widetilde{N}_2$, $Y = N_1N_2+N_1\widetilde{N}_2+\widetilde{N}_1N_2$. There are three possible states for the U-L-D Memory: 1. Up; 2. Down; 3. Lock. If a failure occurred in the Up state, after subtraction of the failing set of increments, it would be desirable to assign weighted probabilities which would be more likely to result in the next state being Down. The probability distributions which can be prepatched are listed in Table I. Likewise, if a success occurs in the Down state, the next assignment of states could be weighted on the same basis.

This strategy can be combined with step size variations implemented with the aid of the success/failure counters.

TESTS

The following tests were carried out using the optimizer in conjunction with ASTRAC I, a ±100 volt, 100 run/sec. iterative differential analyzer.

In order to generate performance measures with the precise characteristics desired, only algebraic functions $F(a_1, a_2)$ were used for quantitative evaluation of the various optimization strategies. For practical use in optimizing dynamic systems, the optimizer setup would be entirely unchanged, except that the function $F(a_1, a_2)$ would be generated as samples at the end of a differential-analyzer run.

Also, the relatively slow repetition rate of 10 runs/sec. was employed only for recording purposes.

Functions Optimized

Initial tests were made with two different types of performance functions.

Minimization of functions of the general type

$$F(a_1, a_2) = \frac{(a_1 - k_1)^2}{a} + \frac{(a_2 - k_2)^2}{b} \quad (6)$$

was carried out. The parameters a_1 and a_2 were allowed to vary over ±100 volts, and convergence of a_1, a_2 was considered to be satisfactory when they were within 0.78 volts of their values which optimized F.

The next experiment involved maximization of a function $F(a_1, a_2)$ exhibiting sharp ridges formed by the intersection of three planes as shown in Fig. 12a. The equation for this function is

$$F(a_1, a_2) = \text{Max} \begin{matrix} a_2 - a_1 \\ -a_2 + 2a_1 \\ -a_2 - 1.5a_1 \end{matrix} + \frac{100}{3} \quad (7)$$

$$a_2 \text{ (optimum)} = 66.6; \; a_1 \text{ (optimum)} = 44.4 \quad (8)$$

Extremely sharp ridges or intersections were simulated by using accurate high-speed selector circuits (Fig. 12b). The parameters a_1 and a_2 were allowed to vary over ±100 volts. Convergence was considered to be satisfactory when F was within 0.5 volt of the maximum.

Strategies Employed

Parameters were allowed to step only after successful runs, i.e., unsuccessful parameter changes were subtracted out.

Various types and degrees of correlation between successive parameter runs were tried. (See Correlated and Biased Random Walk.)

The step size, △, was increased by a factor of 2 after N_S consecutive successes and decreased after N_F consecutive failures; N_S and N_F were varied.

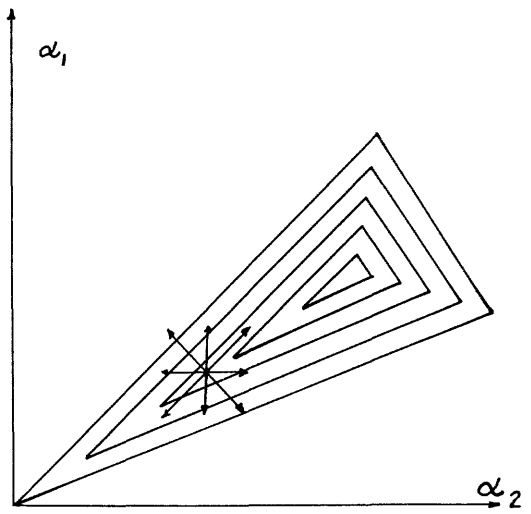


Figure 12a. Ridge.

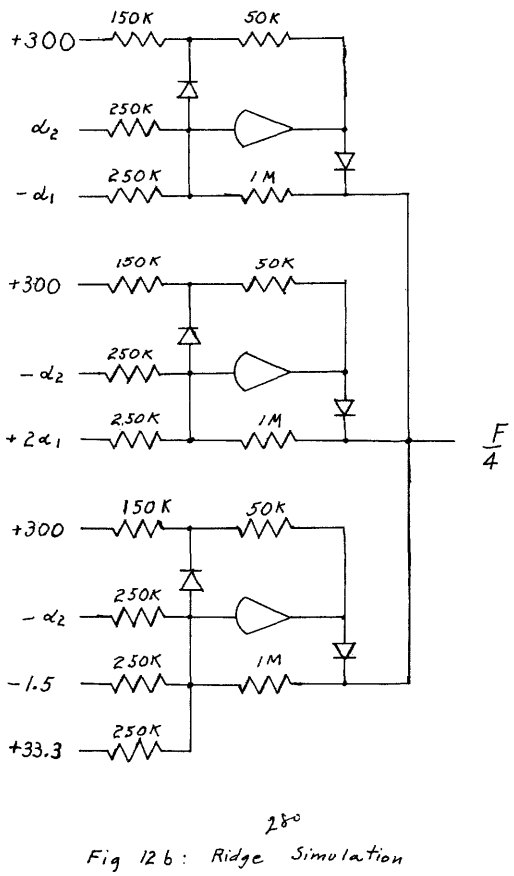


Figure 12b. Ridge Simulation.

Results

For the paraboloids (Equation 6), several thousand optimization trials were recorded using many variations within the general class of strategies outlined above.

As the eccentricity of the contours of constant F was increased, convergence was slowed somewhat but not radically. In no case were more than 70 runs required for convergence.

For the case, a = b = 1, using no correlation in the perturbation scheme, resulted in an average of 28 runs required for convergence with a standard deviation of 9 runs.

For the case, a = b = 1 using strong positive correlation with successes and strong negative correlation after failures, CC = 1; the average trial converged in 16 runs.

The most favorable step-size variation strategy was to increase $\triangle$ after 2 successes and decrease $\triangle$ after 2 failures.

The initial points of $^{\circ}a_1$, $^{\circ}a_2$ were always kept at the extreme of their ranges and only a slight decrease in convergence-time was noted as the initial point was placed closer to the optimum.

The particular ridge (Equation 7; Fig. 12a) to be discussed here is one expressly designed to present the greatest difficulty to the optimizer. Referring to Fig. 12a, it is seen that improvement is quite difficult if the parameter point falls close to the ridge. Had the sides not sloped so steeply, or had the ridge been oriented differently with respect to the parameter axis, the optimizer would have found convergence more natural.

Fig. 13a shows the optimization of this function without step size changes (path A in Fig. 12c). Note that the parameter point followed the direction of greatest improvement until it reached the ridge, then both parameters were forced to zigzag up the ridge to the peak. The step size was constant at 1.56 volts, and 1800 runs were necessary to converge. In Fig. 13b, the step size was again held constant; but the binary counter for a_2 complemented at the be-

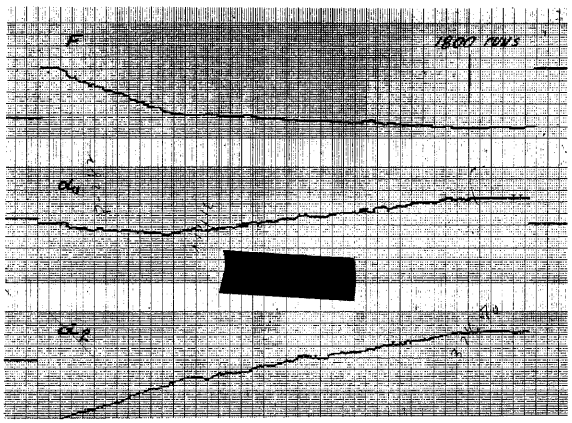


Figure 13a. Optimization of Ridge.

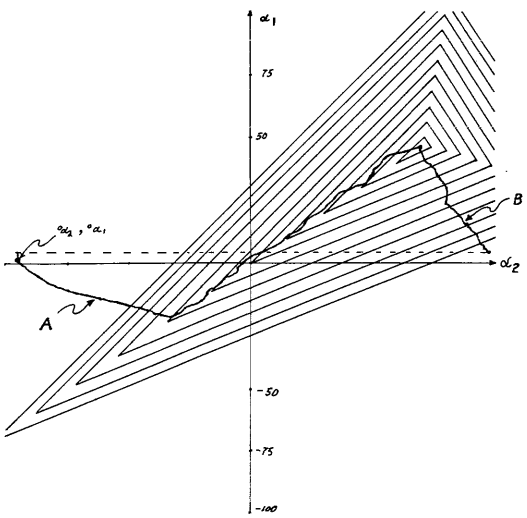


Figure 12c. Optimization Paths.

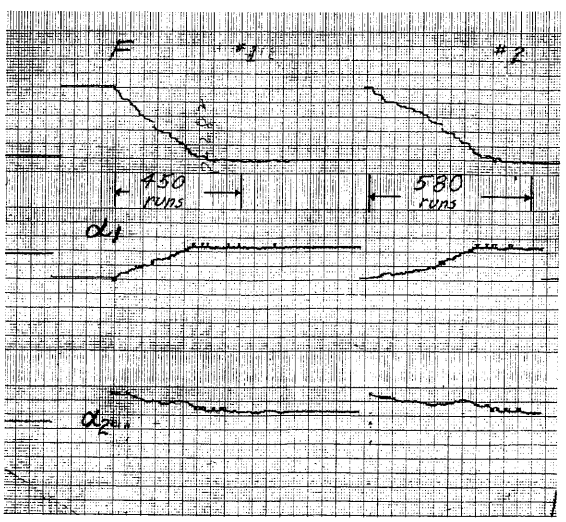


Figure 13b. Optimization of Ridge.

ginning of both trials No. 1 and No. 2. Thus, it was not necessary to climb the longest ridge. The paths converged more quickly up the shorter ridge (path B in Fig. 12c). This favorable possibility is not present in ordinary gradient techniques. Path A in Fig. 12c is most nearly like a path resulting from conventional gradient techniques.

Fig. 14a shows a correlated random search with step-size variations employed in the strategies. The optimizer is not confined to merely zigzagging slowly up the ridge but can traverse great distances while searching for improvements. Correlation improved convergence time somewhat, but only when weak positive correlation after successes was used. For 45 trials using this correlation, the average time to converge was 73 runs if the step-size was decreased after 2 failures. Other types of correlation slowed convergence by 10–20 per cent.

For comparison note that an optimizer going directly in a straight line to the optimum point would require 106 steps if a fixed step size of 1.56 volts were used.

Stopping Conditions

When the optimum value of F is unknown, defining a stopping condition is subject to several factors. One such factor arises when we are not certain that only one peak exists in the domain of F. In this case, we would want to cycle the step size through its decrease-increase scheme several times before stopping. Then a rescaling of the simulation might be desirable in case the initial ranges of the parameters was chosen too large.

Conclusions

While experience with this optimizer is still quite limited, it appears that its performance can compare quite favorably with conventional techniques. Better conclusions can be made when this system is expanded to accommodate four parameters and the logic is enlarged to permit implementation of the conventional deterministic schemes—permitting a direct comparison between random and deterministic methods optimizing the same function.

Presently, however, the random techniques seem well able to handle cases in which the performance measure is not well-behaved (Fig. 3 and 4).

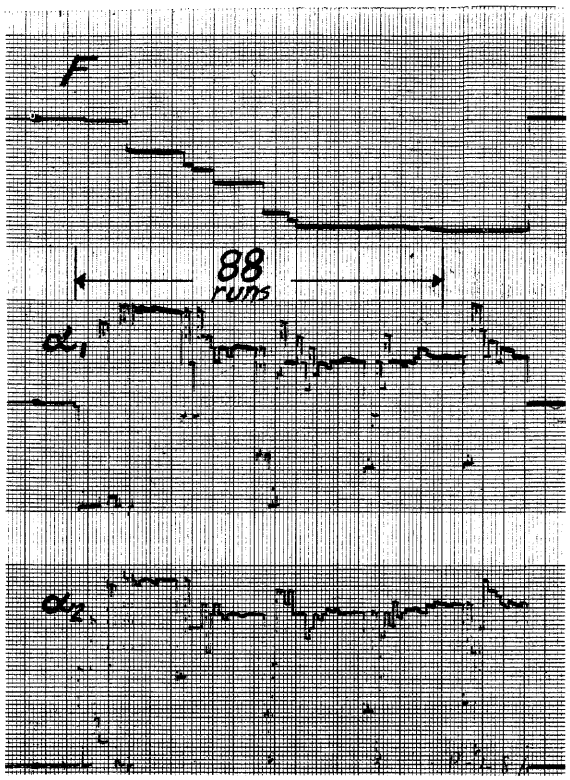


Figure 14a. Optimization of Ridge.

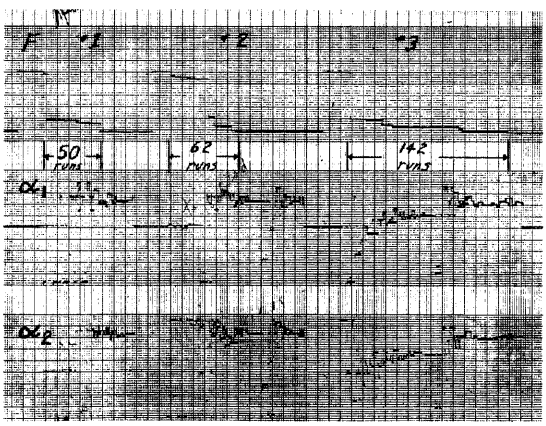


Figure 14b. Employing Step Size Variations.

ACKNOWLEDGEMENT

The project described in this report is part of a hybrid analog-digital computer study directed by Professor G. A. Korn. The writer is grateful to the Office of Aerospace Research, Information Research Division, Air Force Office of

Scientific Research and to the Office of Space Sciences, National Aeronautics and Space Administration for their continuing support of this study under joint grant AF-AFOSR-89-63; and to Professors L. W. Matsch, Dean of Engineering, and H. E. Stewart, Head, Department of Electrical Engineering, for their encouragement and contribution of University facilities.

REFERENCES

1. KORN, G. A., and T. M. KORN, *Electronic Analog and Hybrid Computers,* McGraw-Hill, N. Y., 1964 (in print).
2. MUNSON, J. K., and A. I. RUBIN, "Optimization by Random Search on the Analog Computer," *IRE Trans. PGEC,* June, 1959.
3. WAIT, J. V., and B. A. MITCHELL, "A Simple Solid-State Digital-to-Analog Converter for Hybrid Systems," *ACL Memorandum No. 61,* University of Arizona, 1963.
4. MITCHELL, B. A., "A Hybrid Analog-Digital One Parameter Optimizer," *Ann. AICA,* January, 1964.
5. KARNOPP, R., *Ph.D. Thesis,* Massachusetts Inst. of Technology, 1963.
6. BROOKS, S. H., "A Comparison of Maximum-seeking Methods," *Operations Research,* July-August, 1959.
7. HAMPTON, R., G. A. KORN, and B. A. MITCHELL, "Hybrid Analog-Digital Random-Noise Generation," *IEEE Trans. PGEC,* August, 1963.
8. WITSENHAUSEN, H. S., "Hybrid Techniques Applied to Optimization Problems," *Proc. SJCC,* 1962.
9. HOWELL, M., "Automatic Parameter Optimization as Applied to Transducer Design," *Proc. SJCC,* 1963.
10. HAMPTON, R., "Hybrid Analog-digital Pseudo-random Noise Generator," *Proc. SJCC,* 1964.

APPENDIX I

The following units are contained in the optimizer digital and/or are wired to the optimizer patchbay. On the drawings, a small circle on the end of a wire indicates a patchbay termination.

1. 3 4-bit Gray code counters. One of these counters has a free-running multivibrator for the input. This counter is used exclusively for sequencing operations throughout the logic scheme subroutine, e.g., the analog RESET period. The remaining two Gray code counters have their flip-flop outputs adjacent to two gates on the

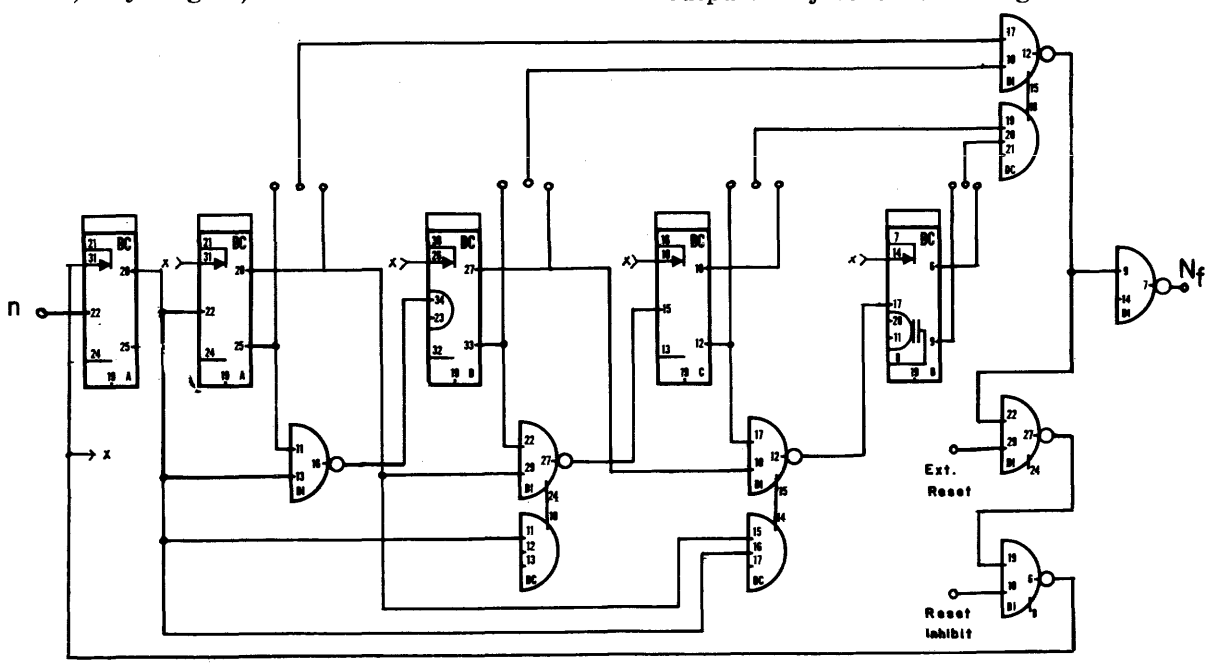


Figure 15. Gray Code Counter Circuit.

patchboard. These may be patched to yield outputs after any preset number of input pulses. These counters may reset themselves or may be reset externally (Fig. 15.)

2. 1 8-bit right-left shift register. The DC set and reset lines, along with the set and reset outputs, appear on the patchboard for parallel drop-in on command. Also the set and reset level controls for the first and last flip-flops are on the patchboard thus permitting operation as a ring counter. Four inputs are provided for shifting in either direction. (Fig. 17b.)
3. 4 8-bit up-down binary counters. The DC set and reset lines, along with the set and reset outputs, appear on the patchboard for parallel loading. A gated complement input for each flip-flop is brought out, allowing the counter to be stepped at any stage. A flip-flop with associated gates permits both pulse and level control of the up-down lines. (Fig. 17a.)

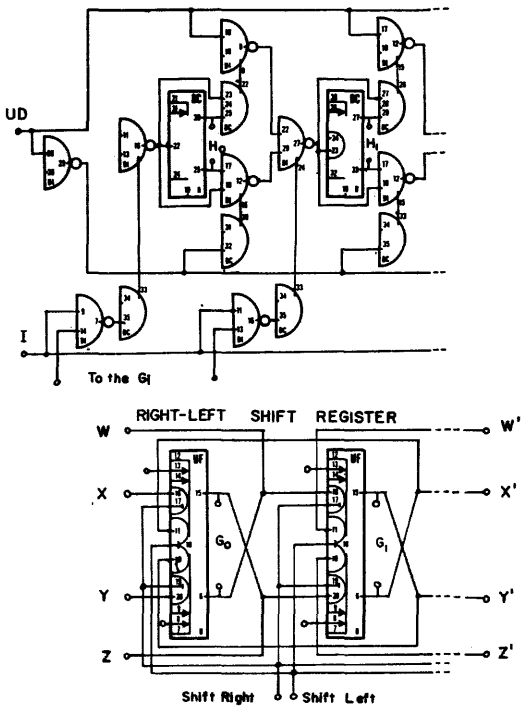


Figure 17a. UP-DOWN Binary Counter.

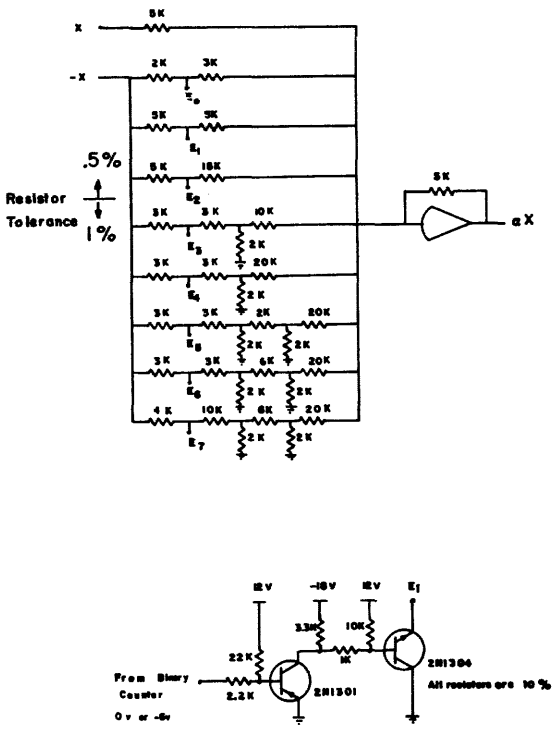


Figure 16. D/A Multiplier and Switch.

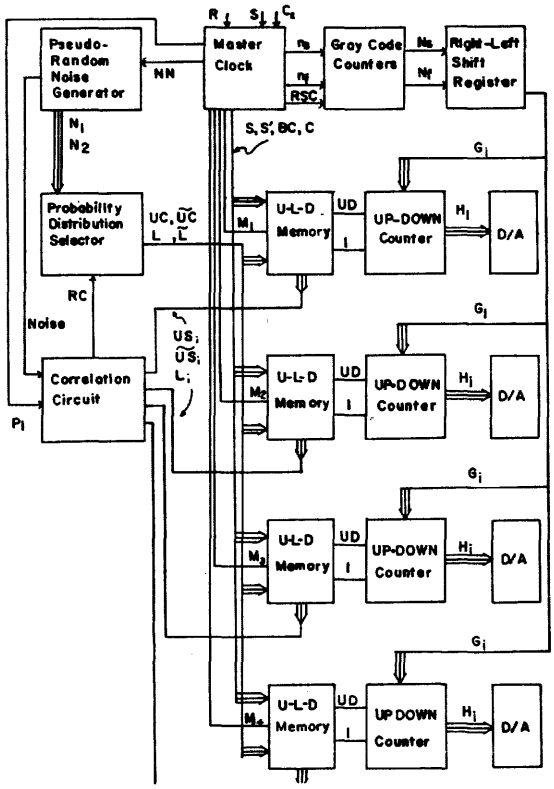


Figure 17b. Right-Left Shift Register.

4. 4 8-bit D/A multipliers. The digital control lines for the D/A multipliers (mounted in shielded cans behind the analog patchbay) are wired to the digital patchbay. Each set of lines is associated with one of the up-down binary counters (Fig 16.)
5. 2 pseudo-random discrete-interval, binary noise generators. The noise generator for ASTRAC II is a 25-stage shift register with modulo 2 adders in feedback, generating a maximum length of $3X10^7$ random bits. This can be divided into two noise generators, each having a $1.5X10^7$ random bit sequence. The noise generator shifts out a random bit when a pulse is applied to the shift input. These inputs, outputs, and complement outputs appear on the optimizer patchbay as well as on the noise generator patchbay.

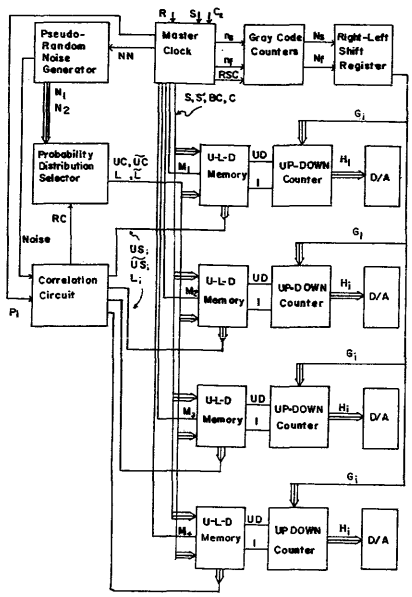


Figure 18. Digital Logic Block Diagram.

A HYBRID ANALOG-DIGITAL PSEUDO-RANDOM NOISE GENERATOR

R. L. T. Hampton
College of Engineering
Department of Electrical Engineering
Analog Hybrid Computer Laboratory
The University of Arizona
Tucson, Arizona

I. INTRODUCTION

Analog and hybrid analog-digital computers used for random-process and statistical studies require noise sources capable of generating random signals whose amplitude distribution, d-c unbalance, spectrum, and RMS level are specified within the computer accuracy limits. Noise samples must not be correlated for time delays exceeding one-ten-thousandth to one-thousandth of a computer run.

Noise derived from thyratrons in magnetic fields, noisy diodes and photomultiplier tubes coated with radioactive paint changes with various environmental conditions. Thus, elaborate gain-control, sampling and/or filter circuitry is required in order to meet computer specifications. By using the random signal from such sources to trigger a flip-flop or threshold-sensing comparator, flat-spectrum binary noise with $\pm A$ volt output levels is obtained. Such binary noise can easily be filtered to produce Gaussian signals and is very useful in its own right: it can be used to drive analog or digital switching systems used to simulate random events, machine failures, etc., and lends itself to direct correlation with other signals without the use of analog multipliers. By precision clamping, the RMS level of binary noise can be closely controlled, but the non-stationarity of the circuits used to obtain electrical noise, even from stationary mechanisms such as a radioactive source, still create problems and expense.[1] For example, the 80 Kc random-telegraph wave generator developed at The University of Arizona's Hybrid Computer Laboratory and described in Ref. 2 required a fairly sophisticated and not completely satisfactory count-rate control loop.

In the design of the University of Arizona's new ASTRAC II iterative differential analyzer, which is to be capable of taking statistics over 1,000 random-input computer runs per second, it was decided to abandon analog noise generation completely. Instead, the machine will employ a digital shift-register sequence generator that can produce binary pseudo-random noise sequences at any clock rate between zero and 4 Mc. This permits exact time scale changes or intermittent operation. The noise generator produces digital computer random numbers as well as analog noise. The digital numbers are easily stored or transmitted and may be used to produce binomially or normally distributed random analog coefficients. Digital multiplexing yields multiple uncorrelated noise signals from a single shift-register. These noise signals are independent of any physical quantity except for the output clamping levels. Also the flat spectrum binary output permits direct logical or analog multiplication. The length of the

pseudo-random output sequence is 33, 554, 431 bits which is equivalent to several thousand computer runs.

II. BINARY PSEUDO-RANDOM NOISE GENERATION WITH SHIFT-REGISTERS

2.1 *Pseudo-Random Binary Sequences*

Binary pseudo-random noise, as the term is used in this paper, differs in two important respects from purely random binary signals:

1. A truly random binary signal is non-periodic, while a pseudo-random sequence repeats itself after some suitably long sequence.
2. In many random binary processes (e.g., a random telegraph wave) the transition from the "1" state to the "0" state (or conversely) can occur at any time, and the state at any *instant* of time is independent of the state at any other *instant* of time. In the pseudo-random binary process the binary level transitions can occur only at specific clock pulse times, separated by intervals during which the binary state is fixed. In this case the state during the fixed time *interval* is independent of the state in neighboring time *intervals.*

A periodic binary sequence will be classified as a pseudo-random sequence if it satisfies the following conditions:

1. In each period the number of "1's" must not exceed the number of "0's" by more than one (or conversely).
2. In each period there must be twice as many sequences of "1's" or "0's" of length n as those of length n + 1.
3. The autocorrelation function must have the form shown in Fig. 3b, i.e., peaked in the middle ($\tau = 0$) and tapering off rapidly at both ends.[3]

Such a binary sequence with a sufficiently long period—longer than a desired series of computer runs—can be used essentially like true random noise for computing purposes.

Periodic binary sequences may be obtained from a digital shift-register with modulo-2 adder feedback (Fig. 1). Module-2 addition generates the sum $(A + B)_{Mod: 2}$ of any two binary inputs A and B according to the following table:

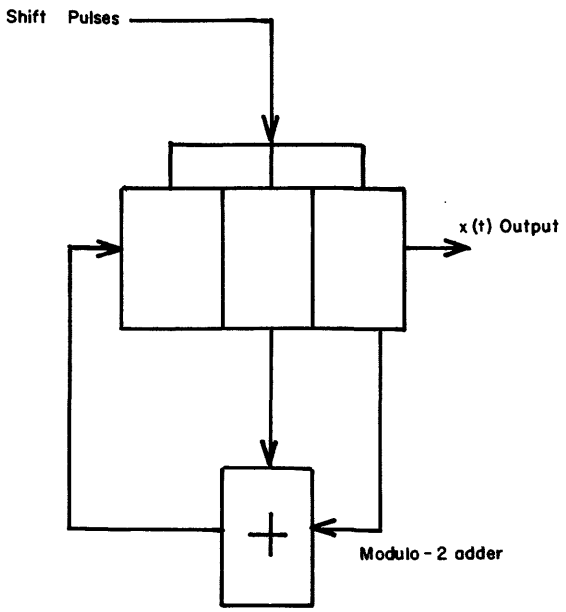


Figure 1.

A	B	$(A + B)_{Mod: 2}$
1	0	1
0	1	1
1	1	0
0	0	0

As can be seen from the above table, modulo-2 addition can be implemented logically with an EXCLUSIVE—OR circuit. The shift-register consists of cascaded flip-flops driven at the desired rate by external clock pulses (shift pulses). The outputs of certain flip-flops are added modulo-2 and their sum is then fed back to the first stage of the shift-register.

Figure 2 illustrates three shift-register periodic sequence generators and their corresponding sequences. Each column of "1's" and "0's" corresponds to the successive states of each stage of the register. It should be noted that No. 1 and No. 2 outputs are periodic every $2^3 - 1 = 7$ bits, while No. 3 is periodic every

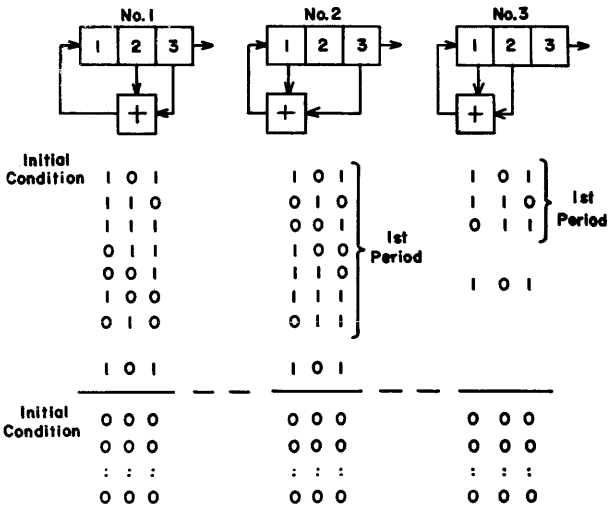


Figure 2. 3-Stage Periodic Binary Sequence Generators.

3 bits. In each case the periodic series is completely determined by the initial state of the flip-flops and by the feedback connections. The resulting sequences for an all-zero initial condition are also shown.

2.2 *Maximum Length Sequences*

It is easy to show that the maximum length of any sequence produced by a shift-register is $2^n - 1$, where n is the number of flip-flop stages. For an n-stage generator, there are 2^n possible states. The all-zero state can be ruled out as an admissible condition, since with modulo-2 addition, each succeeding state would also be all zero as illustrated in Fig. 2. Therefore, the sequence is periodic with a maximum of $2^n - 1$ bits. Golomb has proven that every maximum length shift-register sequence satisfies the three conditions required for a pseudo-random sequence.[3]

The number of different maximum length series obtainable from an n-state shift-register is given by $\frac{\phi(2^n - 1)}{n}$, where ϕ (m) is Euler's Phi Function, defined as the number of integers s such that $0 < s < m$ and s is prime to m. For a three-stage generator $n = 3$, $m = 2^n - 1 = 7$, and the integers s are 1, 2, 3, 4, 5, 6. Therefore, $\frac{\phi(2^3 - 1)}{3} = 2$, so a three stage shift-register can produce two different maximum length sequences, each corresponding to a unique feedback arrangement. Generators No. 1 and No. 2 in Fig. 2 show these two feedback arrangements. Generator No. 3 in the same figure is an example of a feedback arrangement that produces a non-maximum length sequence. Non-maximum length sequences do not in general satisfy the three conditions required for pseudo-random sequences, and thus they are not useful for statistical studies. Consequently, to design a practical pseudo-random noise generator of n-stages, it is necessary to determine the feedback connections which produce a sequence of $2^n - 1$ bits in length.

Another important characteristic of maximum length sequences is the so-called "shift and add" property. That is, when any maximum length sequence is delayed by an integral number of clock periods and then added modulo-2 with the original sequence, a third identical sequence delayed with respect to the first two is formed.[6] This property has a useful application, as will be pointed out in Section III where the actual hardware design of a binary pseudo-random noise generator is considered. The "shift and add" property is also used to derive the autocorrelation function for maximum length sequences (see Appendix).

2.3 *Obtaining the Maximal Period*

A mathematical technique for obtaining maximum length sequences is rigorously presented in Refs. 3 and 5. The method consists of viewing each state of the n-stage shift-register as an n-dimensional vector and the shift-register/modulo-2 adder system as a linear operator, producing the successive states of the n-dimensional vector. Such an operation may be represented by an n x n matrix X. The first row of this matrix corresponds to the first stage of the register, the second row to the second stage, etc. The same is true for the n columns,

i.e., the first column represents the first stage, etc. Each element of the matrix is either a "1" or "0". A "1" in any position indicates that the flip-flop stage denoted by the column drives the flip-flop stage denoted by the particular row. Otherwise, a "0" elements exists. Considering generator No. 1 in Fig. 2, the X matrix may be constructed by observing that stage 1 is fed by stages 2 and 3; therefore in the first row, a "1" is entered in columns two and three. Stage 2 is fed by stage 1, so in row two a "1" is placed in column one. Again stage 3 is driven only by stage 2, hence in row three a "1" is placed in column two. This completes the X matrix for generator No. 1, viz.,

$$X = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

Similarly, for generator No. 2 the defining matrix is

$$X = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

It should be noted that the diagonal below the main diagonal will always consist of a series of "1's", while the first row of the matrix represents the feedback coefficients. Generalizing, the X matrix for n stages may be written:

$$X = \begin{vmatrix} C_1 & C_2 & \cdots & C_i & \cdots & C_n \\ 1 & 0 & \cdots & 0 & \cdots & 0 \\ 0 & 1 & & 0 & \cdots & 0 \\ \cdot & \cdot & \cdot & & & \cdot \\ \cdot & \cdot & & 1 & & \cdot \\ \cdot & \cdot & & & \cdot & \cdot \\ 0 & 0 & \cdots & 0 & 1 & 0 \end{vmatrix}$$

where the C's represent the feedback coefficients ("0" or "1"). The characteristic polynomial of the X matrix is thus,

$$\det [X - \lambda I] \equiv \begin{vmatrix} C_1 - \lambda & C_2 & \cdots & C_i & \cdots & C_n \\ 1 & -\lambda & & 0 & \cdots & 0 \\ 0 & 1 & & -\lambda & \cdots & 0 \\ \cdot & \cdot & \cdot & & & \cdot \\ \cdot & \cdot & & 1 & & \cdot \\ \cdot & \cdot & & & \cdot & \cdot \\ 0 & 0 & \cdots & 0 & 1 & -\lambda \end{vmatrix}$$

$$= (-\lambda)^{n-1}(C_1 - \lambda) - C_2(-\lambda)^{n-2} + C_3(-\lambda)^{n-3} \cdots (-\lambda)^{n-n} C_n$$

$$= (-\lambda)^n \left(1 - \frac{C_1}{\lambda} - \frac{C_2}{\lambda^2} - \frac{C_3}{\lambda^3} - \cdots - \frac{C_n}{\lambda^n}\right)$$

$$= \frac{(-1)^n}{\theta^n}\left[1 - \sum_{i=1}^{n} C_i \theta^i\right]$$

where $\theta = \frac{1}{\lambda}$

The bracketed term $\left[1 - \sum_{i=1}^{n} C_i \theta^i\right] = 0$ is defined as the characteristic equation of the shift-register generator. A necessary, but not sufficient, condition for the period of the shift-register to be of maximum length is that the characteristic equation be irreducible (unfactorable). In Fig. 2 both generators No. 1 and No. 2 have irreducible characteristic equations, but that for No. 3 can be factored. Using a technique known as the "sieve method," the irreducible polynomials which produce maximum length sequence have been tabulated, e.g., see Ref. 4.

2.4 *The Autocorrelation Function*

One of the most important properties of a maximum length sequence generator output is its time autocorrelation function. Since the sequence is periodic every $2^n - 1$ bits, the autocorrelation function will also be periodic every $2^n - 1$ bits. Each bit represents one clock period, $\triangle t$ seconds in duration. In general, the autocorrelation function for the first period of an n-stage shift-register can be shown to be

$$R_{xx}(\tau) = \begin{cases} 1 - |\tau| \dfrac{2^n}{2^n - 1} & \text{for } |\tau| \leq 1 \text{ bit} \\ -\dfrac{1}{2^n - 1} & \text{for } |\tau| \geq 1 \text{ bit} \end{cases}$$

$R_{xx}(\tau)$, for τ equal to an integral number of bit intervals, is derived in the Appendix. Figure 3

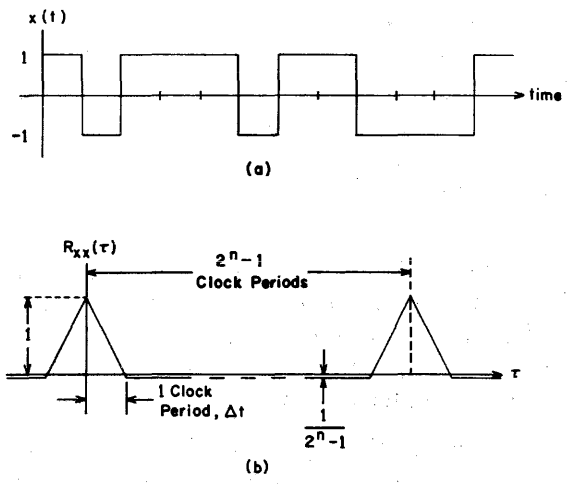


Figure 3.

shows the autocorrelation function of a maximum length sequence x(t). It can be seen that for time delays equal to or greater than one clock period, the correlation is essentially zero for large values of n.

As opposed to the one-peak-per-period autocorrelation function of maximum length sequences, the autocorrelation function of non-maximum length sequences has several positive and negative peaks per period.[10] This results because, as stated before, non-maximum length sequences do not satisfy the three conditions for randomness.

2.5 *The Power Density Spectrum*

From the Wiener--Khintchine theorem, it is known that the power density spectrum $G_{xx}(f)$ of a function x(t) is simply the Fourier transform of the autocorrelation function for x(t). As derived in the Appendix, the power density spectrum for a maximum length pseudo-random binary sequence x(t) is

$$G_{xx}(f) = \frac{1}{Z^2}\,\delta(f) + \sum_{\substack{\alpha=-\infty \\ \alpha \neq 0}}^{\infty} \frac{Z+1}{Z^2} \left[\frac{\sin \alpha\pi/Z}{\alpha\pi/Z}\right]^2 \delta\left(f - \frac{\alpha f_c}{Z}\right)$$

where $Z = 2^n - 1$ = number of bits in one period of x(t)

$f_c = 1/\triangle t$ = clock frequency

From the above expression, it should be noted that

(1) $G_{xx}(f)$ is a discrete spectrum with a harmonic separation of $\frac{f_c}{Z}$ c.p.s.

(2) Nulls occur at integral multiples of the clock frequency f_c.

(3) The bandwidth of $G_{xx}(f) \approx 0.32\, f_c$.

Therefore, for a 25-stage shift-register ($Z = 2^{25} - 1$) and a 4 Mc. clock rate, $G_{xx}(f)$ has a harmonic frequency separation of 0.123 c.p.s. and a bandwidth of 1.28 Mc.

III. DESIGN OF A PSEUDO-RANDOM NOISE GENERATOR

3.1 *Overall Design*

The noise generator employs the same plug-in digital module cards (Computer Control Co. S-Pacs) standard in the new ASTRAC II computer. The block diagram of the entire noise generator system is shown in Fig. 4. This system will operate at clock rates as high as 4 Mc and as low as desired. Multiplexing yields 4 uncorrelated pseudo-random noise outputs from a single 25-stage shift-register. The clamping circuitry produces ±6 volt analog outputs which are easily filtered to generate Gaussian signals; diode function generators can be used to produce other types of analog noise with

specified amplitude distribution and power spectra.

3.2 *The Shift-Register and Modulo-2 Adders*

The shift-register consists of 25 general purpose flip-flops. The SET output of each flip-flop is connected to the Reset Level Control of the next stage, and the RESET output is connected to the Set Level Control (Fig. 5). The shift-register is normally driven from the main 4 Mc ASTRAC II Clock Oscillator. The maximum length period for n = 25 is $2^{25} - 1 = 33,554,431$ bits. Since there are usually many feedback arrangements which will produce a maximal sequence, the one requiring the least hardware will be selected. Using the table in Ref. 4, it was found that if the outputs of stages 3 and 25 are fed back, only one modulo-2 adder is needed. Using this same table, the feedback arrangements, for n = 10 through n = 33, requiring only one modulo-2 adder to produce a maximum length were calculated and are listed in Table 1 of the Appendix. In terms of C.C.C. logic modules, a modulo-2 adder requires 4 NAND gates. The first 3 gates form an EXCLUSIVE—OR, and the fourth gate provides the complement of the EXCLUSIVE—OR circuit (Fig. 5). The output of the modulo-2 adder is fed back to the Reset Level Control of the first shift-register stage, and its complement is fed back to the Set Level Control.

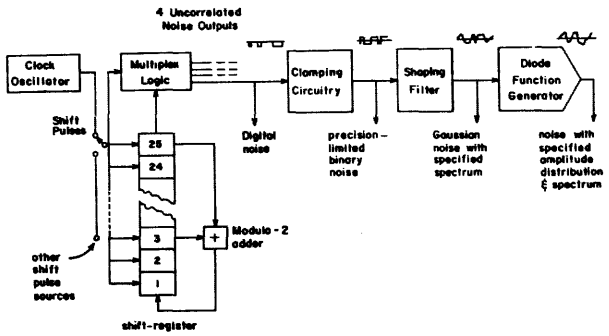


Figure 4. ASTRAC II Noise Generator.

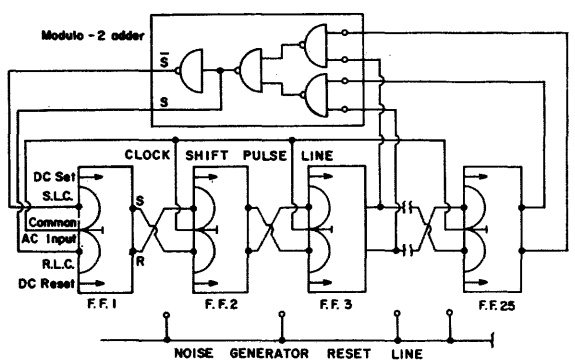


Figure 5. Pseudo-Random Noise Generator Connections.

The output of each flip-flop and its complement are wired to a small patchbay (with a removable patchboard) provided for the noise generator. Since the binary sequence for any stage of the shift-register is the same as that of the preceding stage delayed by one clock interval, the patchbay allows access to 25 sequences—each delayed by one clock interval from the previous one. The DC SET and DC RESET input terminals of each flip-flop are also brought out to the patchbay so that any desired initial state may be patched into the shift-register from the computer NOISE GENERATOR RESET LINE at the start of each computation. This resetting feature is of considerable interest for rechecking a series of Monte Carlo computations with different pseudo-random noise inputs.

If the NOISE GENERATOR RESET patching is omitted, the noise generator will run free and produce periodic sequences at a repetition rate not in general commensurable with the analog computer iteration rate.

To provide the option of having *two completely independent* noise generators, the connections between shift-register stages 11 and 12 are patched rather than wired internally. The first 11 flip-flops require only one modulo-2 adder to produce a maximum length sequence, but the 14 stage shift-register requires three adders to produce a maximal period. One arrangement used to obtain a maximum length sequence from the 14-stage generator is to add the outputs of flip-flops 3, 4, 5, and 14 modulo-2 and to feed the sum back to stage 1.

As one of many alternative patching combinations, two identical 11-stage maximum sequence noise generators, employing only one modulo-2 adder each, may be constructed by using the same feedback connections for the 14-stage shift-register as are used for the 11-

stage register (the outputs of flip-flops 2 and 11 are modulo-2 added and fed back to flip-flop 1). The sequence output of the 14-stage shift-register is then taken from the eleventh stage, leaving 3 stages unused.

3.3 *Multiplex Logic*

The purpose of the multiplex logic is to provide several (in this case four) uncorrelated pseudo-random noise sources from a single shift-register. There are at least two ways of accomplishing this. The first method consists of sampling the shift-register output once every fourth clock interval. This method is used in the noise generator for ASTRAC II, because it requires less digital equipment. The second method utilizes the "shift and add" property of maximum length sequences.

3.3.1 *Method I—Output Sampling*

The output sampling technique uses four staggered 1 Mc pulse trains to sample the 4 Mc pseudo-random noise output. Each pulse train samples the shift-register noise output every fourth clock interval, e.g., the first sampling pulse train senses the state of the noise generator output during the first, fifth, ninth, etc., clock intervals. The final result is four essentially uncorrelated 1 Mc pseudo-random noise outputs, each of which is a maximum length sequence (33, 554, 431 bits if the 25-stage shift-register is used).

Since the 4 Mc noise sequence is sampled only once every fourth clock interval by any one of the 1 Mc pulse trains, it takes 4 periods of the 4 Mc sequence to produce the 33, 554, 431 bit period of any one of the 1 Mc sequences. The 1 Mc sequences do not become periodic after the first 4 Mc period because the number of bits in the 4 Mc period is not evenly divisible by four. In fact, by designating the four 1 Mc outputs as A(t), B(t), C(t), and D(t) respectively, it can be shown (see Fig. 6) that during the second 4 Mc shift-register period A(t) takes the same sequence B(t) had during the first period of the shift-register etc., until the fifth period. Then A(t) begins to repeat the sequence it had during the first shift-register cycle. A similar process holds for the other three sequences B(t), C(t), and D(t). Actually, all four

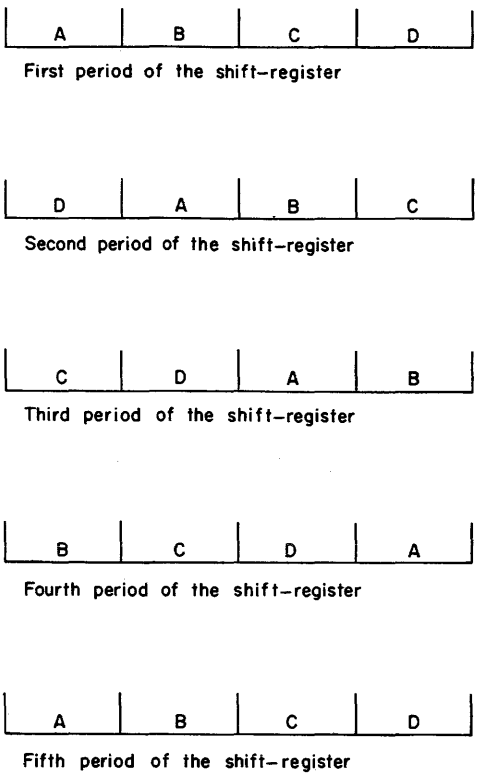


Figure 6.

outputs have the same binary sequence, but each of them is delayed from the previous one by approximately 8, 388, 608 bits. That is, for the outputs A(t), B(t), C(t) and D(t),

$$A(t) = B(t + 8,388,608 \text{ bits})$$
$$A(t) = C(t + 16,777,216 \text{ bits})$$
$$A(t) = D(t + 25,165,824 \text{ bits})$$
$$A(t) = A(t + 33,554,431 \text{ bits})$$

It follows that the cross-correlation function of A(t) and B(t) will peak only after 8, 388, 608 bits. For the sampling rate of 1 Mc, 1 bit is equivalent to 1 microsecond in time. Therefore, the correlation peak occurs approximately 8.4 seconds away from the time origin. The same is true for B(t) and C(t), or C(t) and D(t). *This is long enough for, say, 500 computer runs using 15,000 bits per run from four essentially uncorrelated noise generators.* The digital circuitry required to implement the above operations is shown in the Appendix.

Using the same approach, other combinations are possible. For example, the ASTRAC II

pseudo-random noise generator is designed so two uncorrelated 2 Mc maximum length sequences may be generated. Denoting them E(t) and F(t), then E(t) = F(t + 16, 777, 216 bits). One digital bit in this case is equivalent to ½ of a microsecond. This added feature requires little additional circuitry, as can be seen in the Appendix.

As with the shift-register, the flip-flops of the multiplex logic may be reset as desired by patching between the flip-flop DC SET or DC RESET terminal and the computer NOISE GENERATOR RESET LINE.

3.3.2 *Method II—"Shift and Add" Property*

In effect, the sampling method described above merely produces four identical sequences delayed with respect to one another. In view of the "shift and add" property of maximum length sequences,[6] similar results can be achieved by suitable modulo-2 addition of the various flip-flop outputs. By selecting the arrangement which produces four sequences delayed by approximately 8 million bits from one another, four essentially uncorrelated noise outputs are obtained. The main external difference between this method and the sampling method is that here the noise sequences are still 4 Mc, whereas before they were reduced to 1 Mc.

3.4 *Digital Equipment Requirements*

Using commercially produced (Computer Control Co.) 5 Mc plug-in cards, the entire digital system (25-stage shift-register, 3 modulo-2 adders, and the multiplex logic) costs approximately $2,100. The following components are required:

1. Eight MF-35 flip-flop cards (4 flip-flops per card).
2. Four DI-35 NAND gate cards (8 NAND gates per card).
3. One PN-35 power amplifier card (4 power amplifiers per card).
4. One mounting rack (19 card capacity).
5. Vector patchboard and frame (300 contacts).

The design of the front panel and associated patchboard of the ASTRAC II Noise Generator is shown in Fig. 7.

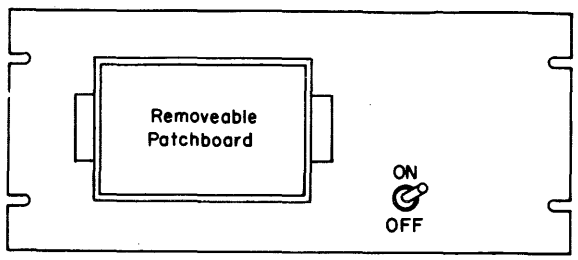


Figure 7. Front View of Noise Generator Panel.

3.5 *Level-Shifting and Clamping Circuitry*

To obtain precision-limited binary sequences of ±6 volts, it is necessary to pass the digital-noise sequences through level shifting and output clamping circuits. The voltage level shift is required since the standard outputs of the C.C.C. digital modules are —6 and 0 volts.

Shown in Fig. 8 is the circuit used to accomplish the level shifting and clamping operations. Transistors Q_1 and Q_2 form a Schmitt trigger. Q_3 provides a current source for the Schmitt trigger, and Q_4 provides a clamped emitter follower type output. The rise time is less than 50 nanoseconds as is the delay through the circuit.

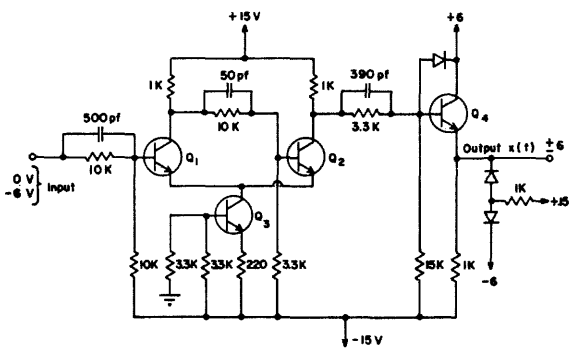


Figure 8. Level-Shifting and Clamping Circuitry.

3.6 *Control of the Mean and Mean Square Value*[11]

3.6.1 *Mean Value*

At high digital-clock frequencies (10 Kc to 4 Mc) used in typical repetitive computer applications (10 cps to 1 Kc repetition rates), the DC level E [x] of the noise generator is best

controlled by a DC blocking filter with the transfer function

$$H(S) = \frac{TS}{1 + TS} \qquad (1)$$

where $T = RC$ (Fig. 9) is an appropriately long time constant. For "slow" analog computation, symmetrical precision clipping of the binary noise output is usually sufficient to control E [x], or the active filter of Fig. 10 may be used to generate the transfer function (1).

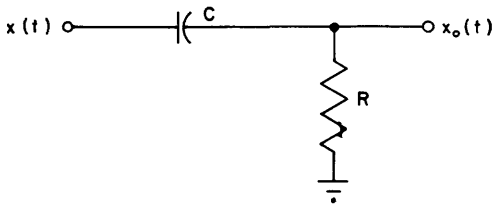


Figure 9. Passive DC Blocking Filter.

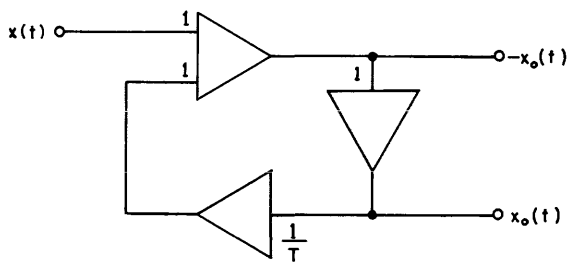


Figure 10. Active DC Blocking Filter.

3.6.2 *Mean Square Value*

Finite time averages of the squared binary noise output x^2 are not subject to fast random variations, and equal one-fourth of the squared peak-to-peak output when a DC blocking filter is used. However, even with a constant load (operational-amplifier input), the peak-to-peak output of the clamped emitter follower in Fig. 8 is subject to slow drift due to temperature variations of the break points of the clamping diodes. For example, a 10° F temperature variation could change the peak-to-peak output as much as 30 millivolts, or 0.25 per cent of the nominal 12 volt value. This corresponds to a 0.5 per cent change in the mean square output E [x^2]. Although drift should be negligible during any one computation, the peak-to-peak voltage will be monitored during computation by oscilloscope comparison with a DC reference voltage read on the computer digital voltmeter. At low clock frequencies, simple analog precision limiters may be used to clip the binary waveform with an accuracy of approximately 0.1 per cent. Still, at high clock frequencies improved transistor clamping circuits, as opposed to more cumbersome AGC-type control, would be useful in regulating the mean square value.

IV. EXPERIMENTAL TESTS

The following experiments are now being conducted with the aid of the ASTRAC I repetitive statistics computer.

4.1 *Time Average and Mean Square Value*

The time average of the pseudo-random noise may be measured by filtering it with a low pass filter. The filter output is then sampled and converted into the appropriate number for a digital counter. The direction of the count is determined by the polarity of the sample. When the preset run counter stops, say, at m computer runs, the number indicated in the counter will be

$$\sum_{k=1}^{m} x_k = m\,\bar{x}$$

Alternately, the time average may be found by using an analog integrator with a long time constant.

By using a diode squaring circuit and the same "sample and count" method used to find the time average, the mean square value is obtained. After m runs the counter will read[8]

$$\sum_{k=1}^{m} x = m\,\overline{x^2}$$

4.2 *Amplitude Distribution*

This measurement is made by using the Amplitude - distribution Analyzer built into ASTRAC I. The analyzer uses a slicer-circuit

which enables its output counter to register one pulse per computer run if and only if the analyzer input voltage x(t) lies between the preset values $\left(X - \frac{\triangle X}{2}\right)$ and $\left(X + \frac{\triangle X}{2}\right)$. The resulting count estimates the Probability that

$$\left[X - \frac{\triangle X}{2} < x\,(t_1) \leq X + \frac{\triangle X}{2}\right].$$

If $\triangle X$ is made sufficiently small, an approximation of the density function is obtained.[9]

4.3 *Correlation Experiments*

A unique method for measuring the autocorrelation function of pseudo-random noise consists of wiring two pseudo-random generators to produce identical maximum length sequences. One of the generators is then driven at a clock frequency of f_1 and the other at a clock frequency of $f_1 + \triangle f$, where $\triangle f$ is very small compared to f_1; thus the noise generator outputs appear with a continuously varying delay with respect to one another. The generator outputs are then multiplied together and low-pass filtered to give the approximate autocorrelation function for direct oscilloscope or recorder presentation.[10]

A more accurate estimation of the autocorrelation function may be obtained by using the ASTRAC I computer to generate the following functions[8]: ${}^kx(t_1) + {}^kx(t_1 + \tau)$ and ${}^kx(t_1) - {}^kx(t_1 + \tau)$, k is the kth computer run and the samples at t_1 and $t_1 + \tau$ are taken during each computer run. The two functions are then square to give

$$[{}^kx(t_1) + {}^kx(t_1 + \tau)]^2 \text{ and}$$
$$[{}^kx(t_1) - {}^kx(t_1 + \tau)]^2$$

After m runs, the final result stored in the counters is

$$\tfrac{1}{4} \sum_{k=1}^{m/2} \left\{ [{}^kx(t_1) + {}^kx(t_1 + \tau)]^2 - [{}^kx(t_1) - {}^kx(t_1 + \tau)]^2 \right\}$$
$$= \sum_{k=1}^{m/2} {}^kx(t_1)\, {}^kx(t_1 + \tau)$$

For a large number of runs,

$$\text{estimated } R_{xx}\,(\tau) = \frac{2}{m} \sum_{k=1}^{m/2} {}^kx(t_1)\, {}^kx(t_1 + \tau)$$

4.4 *Probability Distributions of the Pseudo-Random Sequence*

Assuming the sequence period is large, binomially distributed random numbers q are obtained by counting the noise generator "1's" occurring during Q clock periods.

$$p\,(q) = \left(\frac{Q}{q}\right)\left(\frac{1}{2}\right)^Q \qquad q = 0, 1, 2, \ldots$$
$$E\,\{q\} = Q/2$$
$$\text{Var}\,\{q\} = Q/4$$

Note that, the random variable q is asymptotically normal as $Q \to \infty$. Thus, the random variable u, where

$$u = \frac{q - Q/2}{Q/4} = \frac{2q - Q}{Q}$$

converges in probability to a standardized normal random variable.

V. CONCLUSION

In conclusion, it should be pointed out that such a hybrid noise generator has several striking features not generally shared by existing analog noise generators:

"1. The analog noise output is independent of any physical quantity other than the output clamping levels.
2. The shift-register can be reset at any time to repeat a sequence of random events exactly.
3. The binary noise bandwidth is proportional to the shift register clock rate, which can be changed (and modulated) at will. This permits exact time scale changes or intermittent operation.
4. The noise generator produces digital computer random numbers as well as analog noise. Random digital words are easily stored and transmitted and also may be used to produce binomially distributed random analog coefficients with the aid of D/A-converter multipliers.

5. Digital multiplexing yields multiple uncorrelated noise signals from a single shift-register.
6. The binary output permits direct logical or analog multiplication. In addition, the group-alphabet property of shift-register sequences yields delayed sequences, with the aid of digital logic; which is useful for correlation experiments."[1]
7. The flat spectrum binary noise yields Gaussian noise with any reasonable spectrum by filtering; diode function generators may then be used to produce different amplitude distributions.
8. One noise generator may also be broken into two or more completely independent generators if desired.

In view of the versatility of pseudo-random noise generators and the reliability and decreasing prices of digital-logic modules, the conventional analog noise generator may well become obsolete.

REFERENCES

1. HAMPTON, R., G. A. KORN, and B. MITCHELL, "Hybrid Analog-digital Random-noise Generation," *I.E.E.E. Transactions on Electronic Computers,* August 1963.
2. MANELIS, J. B., "Generating Random Noise with Radioactive Sources," *Electronics,* September 8, 1961.
3. GOLOMB, S. W., *Sequences with Randomness Properties* (Internal Report), Glenn L. Martin Co., Baltimore, Md., June 14, 1955.
4. KEPCKE, J. J., R. M. TAYLOR, and W. S. WYROSTEK, *Geese Techniques for Pseudo Noise Generation,* Defense Systems Dept., General Electric, Syracuse, New York, August 1962.
5. BIRDSALL, T. G., and M. P. RISTENBATT, *Introduction to Linear Shift-Register Generated Sequences,* Technical Report Number 90, University of Michigan Research Institute, Ann Arbor, Michigan, October 1958.
6. PETERSON, W. W., *Error Correcting Codes,* The MIT Press and John Wiley and Sons, Inc., New York, 1961.
7. *Instruction Manual for 5-Mc S-Pac Digital Modules,* Computer Control Company, Inc., Framingham, Mass., July 1962.
8. CONANT, B. K., "A Hybrid Analog-Digital Statistics Computer," *ACL Memo Number 45,* The University of Arizona, Tucson, Arizona.
9. BRUBAKER, T. A., G. A. KORN, "Accurate Amplitude-Distribution Analyzer Combines Analog and Digital Logic," *Rev. Sci. Instruments,* March 1961.
10. STERLING, J. T., *Introduction to Pseudo-Noise Codes and Correlators,* Defense Systems Dept., General Electric, Syracuse, New York, May 1962.
11. KORN, G. A., unpublished lecture notes, University of Arizona, 1963.

ACKNOWLEDGEMENT

The project described in this report is part of a hybrid analog-digital computer study directed by Professor G. A. Korn, who suggested the project and contributed a number of ideas. The author is grateful to the Office of Aerospace Research, Information Research Division, Air Force Office of Scientific Research and to the Office of Space Sciences, National Aeronautics and Space Administration for their support of this study under joint grant AF-AFOSR-89-63; and to Professors L. Matsch, Dean of Engineering and Harry Stewart, Head, Electrical Engineering Department, for their encouragement and contribution of University facilities.

APPENDIX

Multiplex Circuitry

To obtain the four staggered 1 Mc sampling signals, a Gray-Code logic circuit driven by the 4 Mc ASTRAC II clock is used (Fig. A). The Gray-Code effectively divides the clock input by four. The required sampling pulses exist at the outputs of the four NAND gates. They are denoted by ①, ②, ③, and ④ (Fig. B). The multiplexing is accomplished by inverting one of the sampling pulses, for example ④, and applying it to the AC input of a flip-flop. The inversion is necessary for correct triggering. The 4 Mc pseudo-random noise sequence and its

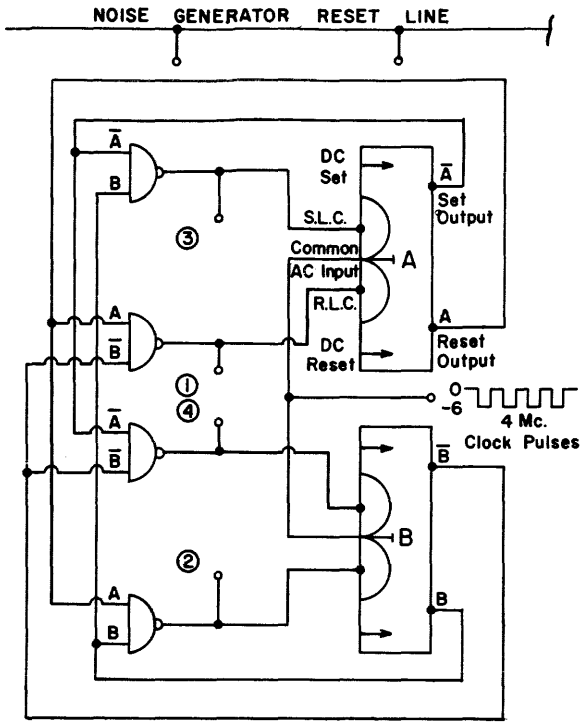


Figure A. Gray-Code Circuit.

complement are applied to the flip-flop Set and Reset Level Controls as shown in Fig. C. The flip-flop output is then the desired 1 Mc pseudo-random noise. The other three 1 Mc noise sources are similarly derived.

Two staggered 2 Mc sampling pulses are easily obtained by NAND-gating pulse trains ① and ② to give the first 2 Mc sampling signal and NAND-gating ③ and ④ to give the second. They are then applied to the AC input of a flip-flop, as was done with the 1 Mc sampling pulses, to obtain two uncorrelated 2 Mc maximum length sequences.

A Gray-Code circuit is used instead of two cascaded flip-flops to provide the "divide by four" operation, because only one of the flip-flops shown in Fig. B changes states per clock pulse. On the other hand, with two cascaded flip-flops, both go through a transition of states every second clock pulse. This would create undesirable coincidence spikes when the cascaded flip-flop outputs were NAND-gated to obtain the four staggered 1 Mc sampling trains.

Derivation of the Autocorrelation Function $R_{xx}(\tau)$ of Maximum Length Sequences[10]

(For τ equal to an integral number of bits or clock periods)

Previously, the pseudo-random noise from a shift-register has been considered to be a se-

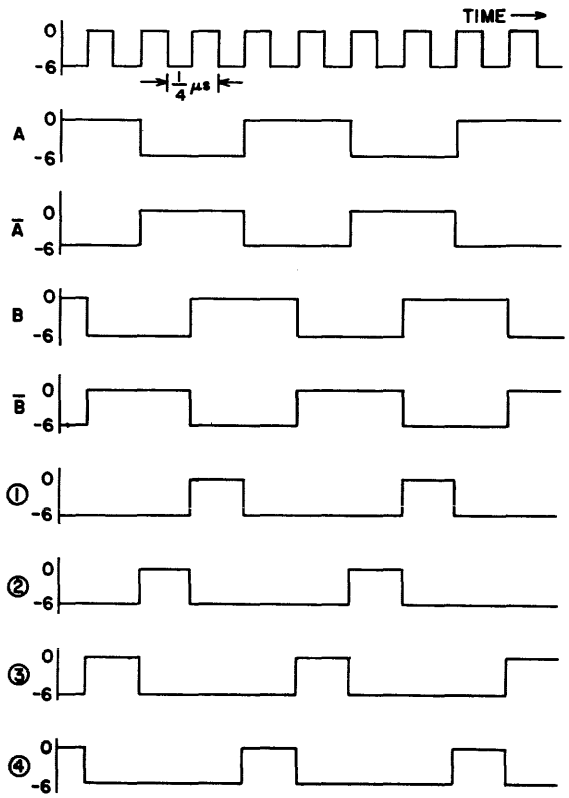


Figure B. Four Staggered 1 Mc Sampling Pulses.

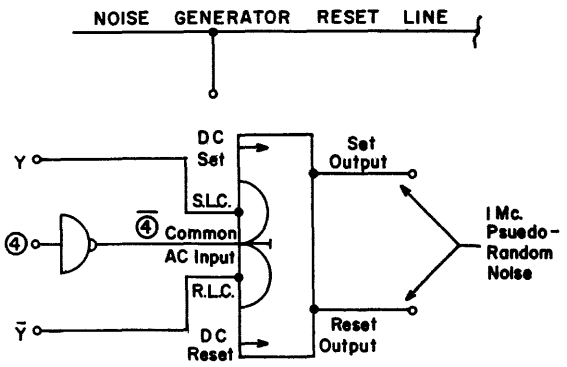


Figure C.

quence of "0's" and "1's", representing the two stable states of the flip-flop stages. If instead, the two stable flip-flop states are designated as "1" and "—1" respectively, it can be shown that modulo-2 addition with "0" and "1" is equivalent to arithmetic multiplication with "1" and "—1".

A	B	(A+B)MOD 2
1	0	1
0	1	1
0	0	0
1	1	0

A	B	A.B
—1	1	—1
1	—1	—1
1	1	1
—1	—1	1

Thus, the "shift and add" property for "1" and "0" sequences becomes the "shift and multiply" property for "—1" and "1" sequences. Since for maximum length sequences of "1's" and "0's", it is true that

$$x(t) + x(t+\tau_1) = x(t+\tau_2),$$

where τ_1 and τ_2 are equal to some integral number of clock periods. It follows that for maximum length sequences of "—1's" and "1's",

$$x(t)\,x(t+\tau_1) = x(t+\tau_2). \qquad (1)$$

The time autocorrelation function for a function $x(t)$ is by definition

$$R_{xx}(\tau_1) = \lim_{T\to\infty} \frac{1}{T} \int_{-T/2}^{T/2} x(t)\,x(t+\tau_1)\,dt. \qquad (2)$$

When $x(t)$ is periodic, as is the case for a maximum length shift-register sequence, no limiting process is needed. Therefore, the autocorrelation equation becomes

$$R_{xx}(\tau_1) = \frac{1}{T} \int_{-T/2}^{T/2} x(t)\,x(t+\tau_1)\,dt. \qquad (3)$$

Since 1 bit is equivalent to 1 clock interval in time, the period $T = 2^n - 1$ bits. Now substituting equation (1) into equation (3) and expressing the limits in terms of n, we have

$$R_{xx}(\tau_1) = \frac{1}{2^n-1} \int_{-\frac{(2^n-1)}{2}}^{\frac{2^n-1}{2}} x(t+\tau_2)\,dt \qquad \tau_1 \neq 0 \qquad (5)$$

$R_{xx}(\tau_1) =$

$\frac{1}{2^n-1}$ times [Number of "1's" in the sequence minus the number of "–1's" in the sequence]

Since the all "1" state (or equivalently, the all "0" state in terms of modulo-2 addition) is a forbidden condition, a maximum length sequence always has one more "—1" state than it does "1" state. Consequently,

$$R_{xx}(\tau_1) = -\frac{1}{2^n-1} \qquad \tau_1 \neq 0 \qquad (6)$$

If $\tau_1 = 0$, the autocorrelation function may be written,

$$R_{xx}(0) = \frac{1}{2^n-1} \int_{-\frac{(2^n-1)}{2}}^{\frac{2^n-1}{2}} x^2(t)\,dt$$

$$= \frac{1}{2^n-1} \int_{-\frac{(2^n-1)}{2}}^{\frac{2^n-1}{2}} 1\,dt$$

$$= 1$$

Derivation of the Power Density Spectrum $G_{xx}(f)$ of Maximum Length Sequences

$$G_{xx}(f) = F\,(R_{xx}(\tau)) = \int_{-\infty}^{\infty} R_{xx}(\tau) e^{-2\pi f\tau}\,d\tau$$

using a Fourier series representation for $R_{xx}(\tau)$

$$R_{xx}(\tau) = \sum_{\alpha=-\infty}^{\infty} r\alpha^{j\alpha\, 2\pi\, f_0 \tau}, r\alpha = \frac{1}{T} \int_{-T/2}^{T/2} R(\tau) e^{-j\alpha\, 2\pi\, f_0 \tau}\, d\tau,\ f_0 = \frac{1}{T}$$

$$G_{xx}(f) = \int_{-\infty}^{\infty} \sum_{\alpha=-\infty}^{\infty} r\alpha e^{j\alpha\, 2\pi\, f_0 \tau}\ e^{-j\, 2\pi f \tau}\, d\tau$$

since the Fourier series for $R_{xx}(\tau)$ converges uniformly, the order of integration and summation can be reversed, giving

$$G_{xx}(f) = \sum_{\alpha=-\infty}^{\infty} r\alpha \int_{-\infty}^{\infty} e^{j\alpha\, 2\pi\, f_0 \tau}\ e^{-j\, 2\pi f \tau}\, d\tau$$

$$= \sum_{\alpha=-\infty}^{\infty} r\alpha\ \delta\ (f - \alpha f_0)$$

$$= \sum_{\alpha=-\infty}^{\infty} \left[\frac{1}{T} \int_{-T/2}^{T/2} R(\tau)\ e^{-j\alpha\, 2\pi\, f_0 \tau}\, d\tau\right]\ \delta(f - \alpha f_0)$$

$$= \frac{1}{T} \int_{-T/2}^{T/2} R(\tau) e^{-j\, 2\pi f \tau}\, d\tau \sum_{\alpha=-\infty}^{\infty} \delta(f - \alpha f_0)$$

$$= \frac{1}{T}\ G_T(f) \sum_{\alpha=-\infty}^{\infty} \delta(f - \alpha f_0)$$

where $G_T(f) = \int_{-T/2}^{T/2} R_{xx}(\tau)\ e^{-j\, 2\pi f \tau}\, d\tau$ is the Fourier transform of $R_{xx}(\tau)$ defined over one period.

$$R_{xx}(\tau) = \begin{cases} -\frac{1}{Z} & T/Z < |\tau| < T/2 \\ 1 - |\tau|\ \frac{(Z+1)}{T} & |\tau| < T/2 \end{cases}$$

where $Z = 2^n - 1$

n = number of shift-register stages.

Evaluating the integral for $G_T(f)$,

$$G_T(f) = \frac{T}{Z^2}(Z+1)\left[\frac{\sin 2\pi f\, T/2Z}{2\pi f\, T/2Z}\right]^2 - \frac{T}{Z}\left[\frac{\sin 2\pi f\, T/2}{2\pi f\, T/2}\right]$$

$$\therefore G_{xx}(f) = \frac{1}{T}\sum_{\alpha=-\infty}^{\infty}\left(\frac{T}{Z^2}(Z+1)\left[\frac{\sin 2\pi\alpha f_o\, T/2Z}{2\pi\alpha f_o\, T/2Z}\right]^2 - \frac{T}{Z}\left[\frac{\sin 2\pi\alpha f_o\, T/2}{2\pi\alpha f_o\, T/2}\right]\right)\delta(f-\alpha f_0)$$

$$= \sum_{\alpha=-\infty}^{\infty}\left(\frac{Z+1}{Z2}\left[\frac{\sin\alpha\pi/Z}{\alpha\pi/Z}\right]^2 - \frac{1}{Z}\left[\frac{\sin\alpha\pi}{\alpha\pi}\right]\right)\delta(f-\alpha f_0)$$

$$= \frac{1}{Z^2}\delta(f) + \sum_{\substack{\alpha=-\infty \\ \alpha\neq 0}}^{\infty}\frac{Z+1}{Z^2}\left[\frac{\sin\alpha\pi/Z}{\alpha\pi/Z}\right]^2\delta\left(f-\frac{\alpha f_c}{Z}\right)$$

where f_c = clock frequency = Zf_0.

n	Add Modulo-2 the Outputs of Stages
10	3,10
11	2,11
15	1,15
17	3,17
18	7,18
20	3,20
21	2,21
22	1,22
23	5,23
25	3,25
28	3,28
31	3,31
33	13,33

(n equals the number of stages)

Those values of n between 10 and 33 not appearing above can not produce a maximum length sequence with one modulo-2 adder.

FEEDBACK CONNECTIONS REQUIRING ONLY ONE MODULO-2 ADDER TO PRODUCE A MAXIMUM LENGTH SEQUENCE: n = 10 through n = 33

TABLE 1

A DELTA-SIGMA MODULATION SYSTEM FOR TIME DELAY AND ANALOG FUNCTION STORAGE

H. Handler
Instructor, Electrical Engineering Department
University of Arizona
Tucson, Arizona
and
R. H. Mangels
Research Assistant, Electrical Engineering Dept.
University of Arizona

INTRODUCTION

The advent of fast iterative analog computers[1,2,3] poses the requirement for storing analog functions for integral multiples of the computer repetition period which is typically of the order of 1 to 10 milliseconds. In addition to function storage for one computer run, it would be desirable to store analog functions for indefinite time intervals to permit table lookup. Typical accuracy requirements might be 0.1 to 0.5 per cent of half-scale at ten times the computer repetition frequency (10 cps to 1000 cps) with somewhat lower accuracy at higher frequencies.[4]

For most iterative-differential-analyzer applications, existing function storage schemes, such as capacitor wheels, reed-relay matrices, cathode-ray storage tubes, storage integrators, lumped-parameter operational amplifier schemes, and magnetic recording techniques [5,6] all distinguish themselves by varying degrees of impracticality, high cost and low accuracy.

Magnetostrictive (acoustic) delay lines have been used successfully in digital computers.[7] These lines are highly developed and widely available in packages which include all write and read circuitry. Bit rates up to 5 Mc with several milliseconds of delay are possible. Pulse regeneration with clock-gated logic permits cascading of delay line sections and indefinite recirculation of digital data.

The magnetostrictive delay line is for all practical purposes designed for digital pulsed signals. Consequently, in order to use this type of line it is necessary to encode the signal into a pulse modulated form. Pulse amplitude modulation is unsuitable because attenuation and reflection in the line result in poor signal to noise ratios. Since pulse frequency modulation and pulse position modulation require precise timing circuits to effect pulse regeneration, it would seem practically impossible to use these schemes for indefinite recirculation of analog stored data.

True pulse code modulation (PCM) is without question the most efficient modulation scheme. PCM will, however, require a complete set of A to D and D to A converters to achieve modulation and demodulation. This method is used in the Electronics Associates HYDAC computer[8] but true PCM seems prohibitively expensive for all but the largest computer installations.

Delta modulation,[9] i.e., pulse modulation through suppression of pulses in a constant

frequency clock pulse train, appears especially attractive, for it combines inexpensive modulation and demodulation with the possibility of pulse regeneration with clock gated logic just like true PCM. Conventional delta modulation which transmits the time derivative of an analog input is suitable for voice communications, but not for transmission of absolute d-c levels.

Inose et. al.[10] circumvented this drawback to delta modulation by modifying the transmitter so that the pulse train represents information about the signal rather than its derivative. This modified delta modulation or delta sigma modulation meets the requirements for an analog signal encoder which can handle signals extending down to zero frequency. The modification has not altered the intrinsic simplicity of the delta modulation system making it very attractive for medium size computer installations.

CIRCUIT OPERATION

Referring to the block diagram of Fig. 1, the input signal is summed into an integrator along with the flip-flop output signal, $-F(t)$. The integrator output sets a two-level comparator which then determines whether a gating circuit will allow a set or reset pulse to pass. If the integrated sum of the input and output signal is less than zero at the "sampling" time a reset pulse is gated out. If the amplitude of the integrated sum is greater than zero a set pulse is released to the flip-flop.

The waveforms which appear in the system for a zero input signal are shown in Fig. 2. An

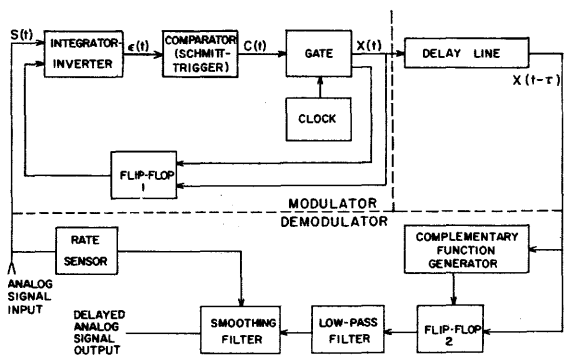


Figure 1. Block Diagram of Analog Delay Unit.

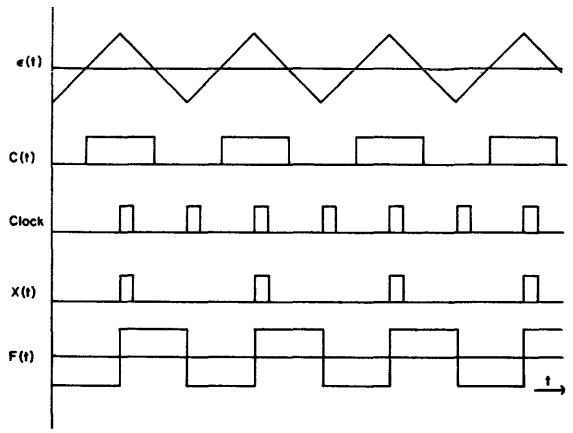


Figure 2. Modulator Waveforms for Zero Input Signal.

examination of these waveforms will clarify the behavior of the system. If the transmitter is in a non-saturated condition, the feedback loop will attempt to keep the integrator output close to zero. Under these assumed conditions the integrated difference between the signal and the flip-flop waveform will be less than some arbitrary number ϵ, or

$$\int_0^t \Big(S(t) - F(t)\Big)dt \leq \epsilon(t) \qquad (1)$$

If one considers the quantity ϵ to be small then the flip-flop output $F(t)$ is approximately equal to $S(t)$ in an average value sense. Therefore, to recover the modulating signal it is only necessary to insert the flip-flop output into a low pass filter; the output of the Schmitt trigger may also be used for this purpose. As used for computer memory, gated clock pulses taken from one of the feedback lines to the flip-flop are sent down the delay line and used to operate another flip-flop in the receiver rather than attempting to transmit and reshape the wider pulses occurring in the flip-flop's output.

Five megacycle Computer Control Corp. (CCC) transistor logic cards were used for the digital logic along with Burr-Brown ± 10 v operational Type 1607A amplifiers. The demodulator is a passive RLC filter of the Tchebysheff II type[11] designed for 50 kc cutoff and preceded by a wideband operational amp. (10 mc 0-db

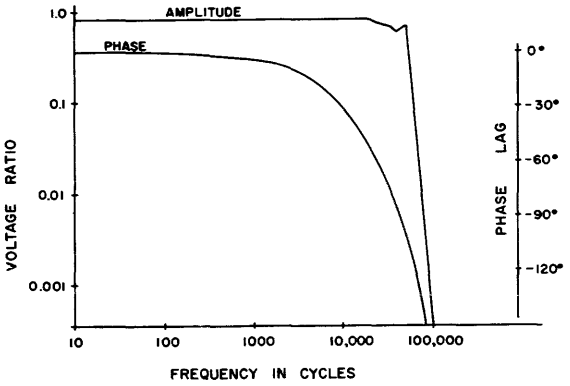


Figure 3. Demodulating Filter Response.

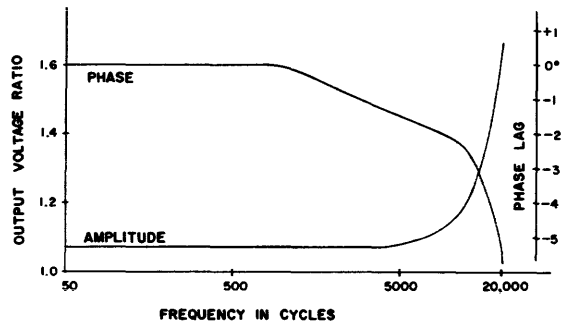


Figure 4. System Response of Complete $\Delta\Sigma$ System.

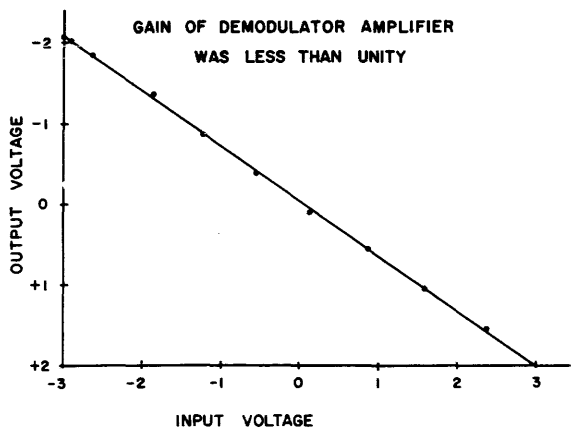


Figure 5. Static System Response.

bandwidth). Filter cutoff at 50 kc seems to be about optimum for a 2 mc bit rate; values above this admit excessive noise into the output. The demodulator has objectional phase shift above 1 kc (see Fig. 3 for filter response, which was improved, at the expense of some amplitude response, by lead networks at the input, Fig 6.) The overall system response is shown in Fig. 4. The uncompensated system is flat out to 40 kc. Fig. 5 shows the static input output characteristic. This curve was obtained with a digital voltmeter and is seen to be quite linear.

The output pulse widths of the CCC Delay Multivibrator Cards were narrowed to about half their normal minimum value. It was found necessary to add several additional logic cards to what could be considered a minimum component delta sigma system in order to prevent cross coupling effects from causing pulse-train jitter. Such jitter makes pulse reshaping more difficult, but has no noticeable effect on other aspects of system performance.

Operation of the modular circuit, shown in Fig. 6, is straightforward. The 2 mc clock trig-

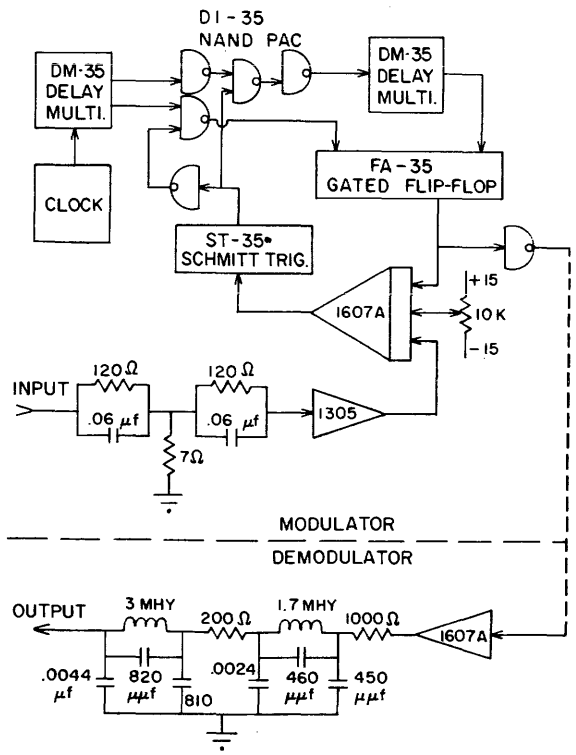


Figure 6. $\Delta\Sigma$ Modulator Demodulator.

gers a delay multivibrator, whose complementary output pulses are NAND gated so that one or the other is allowed to reach flip-flop FF1. The flip-flop output is added to the analog input, and the sum is integrated. The integrated output is fed into a Schmitt trigger which in turn controls the NAND gates in such a manner that feedback of the flip-flop output always tends to return the integrator output to zero. As a result, the running average of the output pulse F(t) is proportional to the analog input to the integrator.

SYSTEM RESPONSE

Although it is difficult to determine the modulator response or pulse code pattern for an arbitrary input, it is instructive to consider the system response to selected inputs.

As Fig. 2 indicates, the flip-flop output for a zero input signal consists of a square wave with the transitions occurring at the clock intervals. Consider an input step voltage equal to one half the flip-flop voltage. The flip-flop pulse pattern which would result from this input is displayed in Fig. 7 along with various other waveforms in the system. The average value of the flip-flop voltage F(t) is $E_o/2$, one half its peak value where E_o is the amplitude of the flip-flop output. The fundamental frequency of the waveform for this input is 1/4T where T is the clock period. If this waveform is averaged by a low pass filter, the steady state output of the filter would be E/2 volts or the input signal.

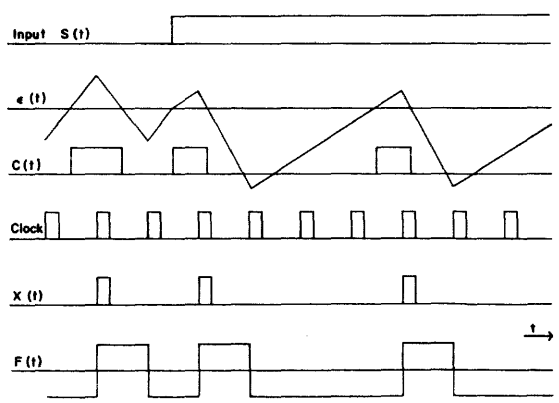


Figure 7. Modulator Waveforms for Half Full Scale Input.

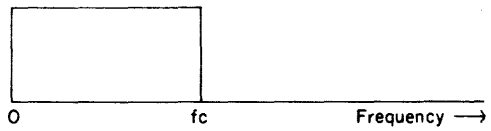


Figure 8. Ideal Low-pass Filter.

If the system is to be responsive to step voltages in a reasonable amount of time, it is necessary to keep the bandpass of the averaging filter as large as possible. On the other hand, too large a bandpass would permit an unacceptable amount of ripple to appear at the output. If one assumes the existence of an ideal low pass filter as shown in Fig. 8 and if the cutoff frequency f_c is set below 1/4T cycles per second, no ripple components will appear in the steady state output. Unfortunately, the ripple frequency is a function of the input level and for inputs close to $\pm E_o$ volts, the fundamental frequency of the flip-flop waveform becomes arbitrarily small. In fact, for input levels equal to $\pm E_o$ the system reaches the limits of its dynamic range and the flip-flop remains permanently in one state. For an averaging filter with a bandpass f_c cycles per second the static range of the input must be limited to[10]

$$(f_cT)\ E_o < E_{in} < (1—f_cT)\ E_o. \qquad (2)$$

In a dynamic situation, the input signal may have a maximum amplitude larger than E_o volts. The stable pulse pattern responsible for Equation 2 does not result unless the level remains constant for periods of time commensurate with the averaging bandwidth, thus this dynamic range constraint does not hold for time varying signals.

Triangle and square waveform response of this system were checked from .01 to 1200 cps. Amplitude response is linear and once the input and output are superimposed on a dual trace scope they remain superimposed as the input amplitude is varied throughout the system range. Waveform response for a 2 mc bit rate is shown in Figs. 10a through f. Input compensation was removed since it is not correct when a variable bandpass filter is used.

Triangle waveform response at 500 cycles along with the modulating signal is shown in Figs. 9a and b for two values of demodulating

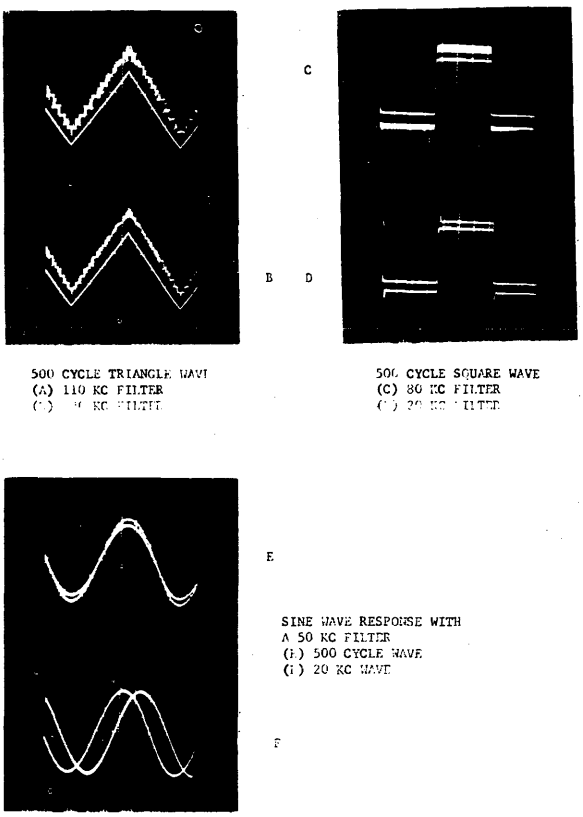


Figure 9. Waveform Response of $\triangle\Sigma$ System.

filter cutoff. Fig. 9a shows a wide filter setting of 110 kc in order to show quantizing effects of the system. Figure 9b shows a more normal filter setting of 20 kc which filters out much of the noise and effectively recovers the modulating signal. Square waveform response at 500 cycles is shown in Figs. 9c and d for two values of demodulating filter cutoff frequency. Figure 9c is for a 80 kc filter and Fig. 9d for a 20 kc filter. The only effect of filter width on a square waveform is in the height of the grass on the horizontal segments. The extra height hides the overshoot in the wide filter. Note that there is no measurable difference in the rise time of the input and output and that the signal response is only dependent upon the characteristics of the demodulating filter.

Sine wave response was checked from .01 cycles up to 600 kc.[12] While the system will still respond at these high frequencies, noise content becomes objectionable above 100 kc and sine wave response was not considered useful above this frequency. At frequencies below 50 kc amplitude response is excellent with the same general properties commented on for triangle and square waves at lower frequencies. Sine wave response is shown in Figs. 9e and f for a 50 kc filter and a 500 cycle and 20 kc sine wave respectively. In Fig. 9e the input is the lower amplitude wave while in Fig. 9f it is the wave occurring first in phase. At 20 kc the demodulating filter phase shift is quite noticeable. As predicted in the S/N ratio section there is less noise associated with the high frequency waveform which is close to the filter cutoff frequency.

ADAPTIVE FILTERING

It is apparent that the requirements for good noise suppression and high frequency response do not coincide. However, it is possible to vary the bandwidth of the low pass filter, leaving it wide for rapidly varying signals and decreasing it for low bandwidth signals.

Since the demodulator or low pass filter is on the receiving end of the delay line, the shape of the signal is known five milliseconds before the pulse code reaches the filter. It is, therefore, possible to examine the signal at the input of the modulator and insert either a wide or narrow bandwidth demodulator depending on the nature of the input. The rate sensing circuit (Fig. 10) consists of an R-C differentiator feeding an absolute value circuit which sets a two level comparator. The demodulator then acts as an adaptive filter whose bandwidth is varied commensurate with the requirements of the input signal.

DYNAMIC RANGE

The system exhibits little, if any, threshold effect on input level below which the system does not operate. Measurable amounts of thres-

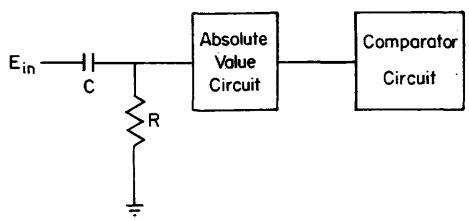


Figure 10. Block Diagram for Demodulating Filter.

hold can be introduced if the integrator capacitor is bypassed with a resistor; the amount of the threshold increases as the integrator time constant is decreased. Decreasing inputs in the unbypassed system eventually becomes lost in the system output noise at about —40 db below saturation. This figure can be improved by sufficiently narrowing the bandpass filter. With a narrow filter, 200 cps, inputs ranging from 8v down to 5 mv or over a 64 db dynamic range can be detected. The dc level adjustment on the integrator input was always set to center the input in the dynamic range, i.e., saturation occurs at the same level for positive and negative excursions. This is the same dc level which gives an alternating on-off pulse pattern at the output for zero input, as shown in Fig. 2.

OUTPUT SIGNAL TO NOISE RATIO

Delta sigma modulation is a pulse code modulation system and like true PCM systems, is subject to quantizing noise. Fig. 9a is a photograph showing the output signal for a triangle wave input. The quantizing phenomenon is quite apparent from the photograph. Inose,[10] using a somewhat simplified model, derives an expression for the signal to quantizing noise ratio as:

$$\frac{P_s}{P_N} = M \frac{3}{4\pi} \frac{f_r}{f_c}^{3/2} \qquad (3)$$

where M is the ratio of the signal amplitude to the dynamic range, f_r the clock frequency and f_c the cutoff frequency of the low pass filter.

There is one significant respect in which the quantizing phenomenon of a delta sigma system differs considerably from that of a true PCM system. If the input signal is a constant d-c level, the modulator feedback will adjust the pulse rate so that the average value of the flip-flop waveform F(t) equals the input voltage. Hence, no measurable quantizing errors are encountered in the transmission of d-c levels. On the other hand a conventional PCM system initially quantizes the input signal, and the maximum error which may occur at this point is $a/2$ where a denotes the voltage of a quantizing interval.

Signal to noise ratio was measured as a function of bit rate, input amplitude, filter cutoff frequency, modulating frequency and integrator time constant. The results are plotted in Fig. 12 against the theoretical Equation 3 as shown in Fig. 11a. The normal fixed low-pass de-

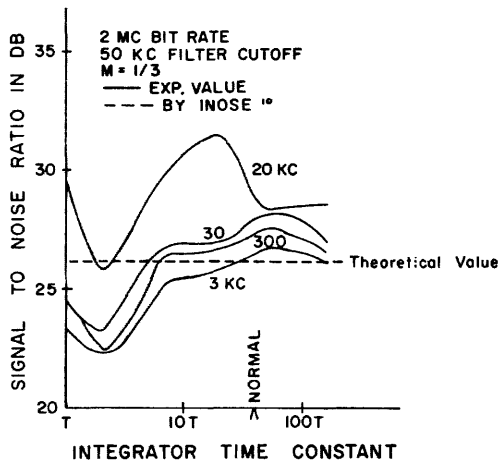


Figure 12a. △Σ S/N Ratio Using Four Modulating Frequencies.

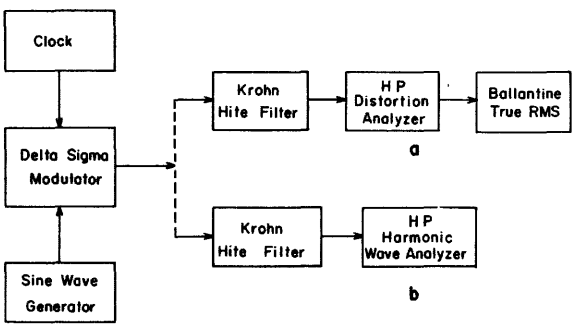


Figure 11. Equipment Setup for Measurement of Signal to Noise Ratio and Spectrum.

modulating filter was replaced by an adjustable bandpass filter for use in the spectrum analysis. This was an active Kron Hite which has a cutoff of 24 db per octave and was modified to have a fixed low frequency cutoff of about 5 cycles. The HP Distortion Analyzer contains an 80 db rejection filter. The filter output was read on a true rms Ballantine voltmeter for accurate power evaluation. The measurement technique was to adjust the distortion analyzer to a constant output level on the Ballantine, tune out the fundamental, measure the new output and

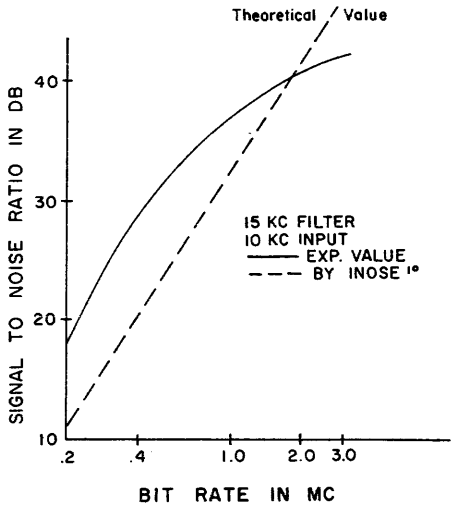


Figure 12b. $\Delta\Sigma$ S/N Ratio as a Function of Bit Rate.

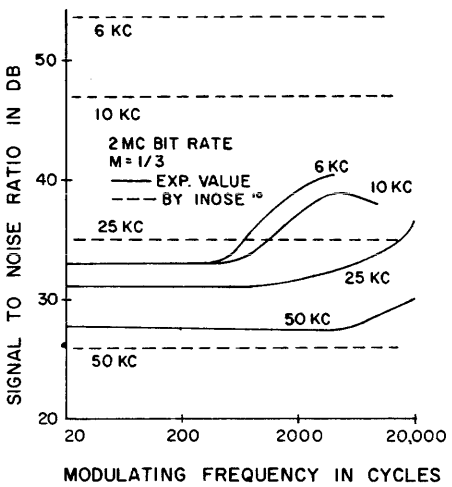


Figure 12c. $\Delta\Sigma$ S/N Ratio Using Four Filter Cutoff Frequencies.

calculate the S/N ratio. The distortion analyzer has a bandwidth of 80 kc which had negligible effect on the S/N ratio of test signals.

The exact value of M for this system is difficult to determine since the modulator saturates slowly and some judgment must be used in evaluating the maximum amplitude. Also most of the distortion resulting from mild saturation levels can be removed by sufficiently narrowing the demodulating filter. Saturation for these measurements was determined as follows. Demodulating filter cutoff was set to one tenth the bit rate and its output observed on a scope. The input level was reduced from obvious saturation until a continuous grass just formed across the top of the demodulated output. Using this method, M was found to be independent of bit rate, input frequency and of course filter cutoff. Below a minimum value of integrator time constant, about 3T or 150 nsec (Fig. 12a) the saturation level decreases as T decreases, otherwise M is also independent of T; this arae in addition shows a poor S/N ratio. All other tests were performed at a time constant of 40T.

It is noted that in most instances the measured value of the S/N ratio is higher than the

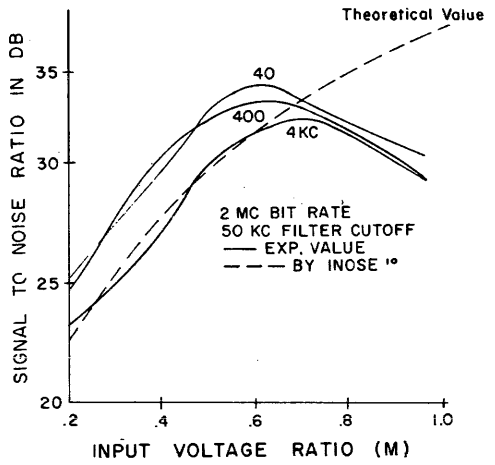


Figure 12d. $\Delta\Sigma$ S/N Ratio Using Three Modulating Frequencies.

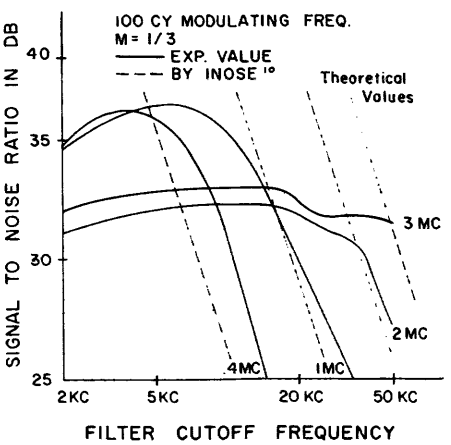


Figure 12e. $\Delta\Sigma$ S/N Ratio Using Four Bit Rates.

theoretical prediction of Equation 3. For Fig. 12b the values chosen were such as to be proportional to a similar plot given in Reference 10 for which the experimental curve has the same shape, but is a few db below theoretical value.

It is noted in Fig. 12c that the S/N ratio, as predicted by Equation 3, is independent of modulating frequency except at the high end of the modulating range. In all cases the S/N ratio increases as the modulating frequency approaches within four octaves of the filter cutoff frequency. This is not predicted by Equation 3 but is anticipated from spectrum studies. Since the modulating frequency is always the lowest spectrum line more harmonics will be removed from the total noise content of the output as the fundamental frequency approaches the filter cutoff frequency. For a perfect filter only the fundamental and noise would be left when the fundamental is within one octave of the cutoff frequency. While the noise power is present across the entire filter bandpass, its level is very low for high bit rates (see Figs. 13d, e, f).

S/N ratio as a function of M is shown in Fig. 12d. It is noted that the theoretical and experimental curves for low M follow quite closely over a wide range of modulating frequencies, falling off only above .6M. This result is derived theoretically [13] by consideration of a simplified modulating system so constructed that it may be analyzed mathematically. Simplification is achieved by the elimination of the flip-flop from the feedback loop of the delta sigma system and removing one of the feedback lines. This results in the gated clock pulses being fed directly to the integrator input. The output of the integrator then becomes a series of rectangular pulses which can be expressed as a Fourier transform and multiplied by the demodulating filter frequency response function. Theoretical results obtained from this simplified system parallel quite closely the experimental results obtained from the Delta-Sigma system.

Figure 12e shows the effect of varying the filter cutoff. While the theoretical and experimental curves tend to compare favorably at high cutoff frequencies, the S/N ratio falls off instead of continuing upward with continued decrease of the filter cutoff frequency as predicted by Equation 3. Figure 12c also illustrates this point for the 2 mc bit rate.

PULSE-TRAIN SPECTRA AND FREQUENCY RESPONSE

Spectrum studies were conducted on the flip-flop output. The test setup is shown in Fig. 11b. It was found necessary to impose a filter whose bandpass was slightly wider than the analyzer in order to prevent mixing of the spectrum at the analyzer input. The test procedure was simply to select a bit rate and sinusoidal modulating frequency and to measure the amplitude and frequency of the spectrum components at the filter output. The ratio of input voltage to maximum input, M, as given in Equation 3 was maintained at about 0.8.

Figures 13a to l shows pulse-train spectra for sinusoidal inputs plotted with frequency in kc and amplitude in mv. Besides the input frequency and some of its harmonics, other frequency components can be seen. The first few harmonics of the input frequency are often missing. As an example, with a 50 kc bit rate and a 20 kc filter cutoff the spectra for 1, 2, and 4 kc inputs are shown in Figs. 13a, b, and c, respectively; spectra are plotted up to 16 kc, the analyzer limit. Note that the second harmonic is always missing. In Fig. 13a, most of the higher harmonics reappear, starting with the eighth. In Fig. 13b all harmonics above the first are suppressed with the appearance of a new frequency component halfway between each harmonic location. In Fig. 13c, the higher harmonics are present along with a new frequency component which is one quarter of the way between harmonics. A continuous noise spectrum is present for all but zero input; the noise level is a function of both modulating level and input frequency. This is assumed to be the result of intermodulation around the loop since there is no measurable output noise for zero input. The amplitude of the fundamental was always exactly proportional to the input level, but the other spectrum lines increase and decrease as the input level is increased; a typical result is shown in Fig. 14 for the spectrum in Fig. 13b. Fig. 14a shows the amplitude of the fundamental frequency of 2 kc, which is

seen to be quite linear with input level, plus the next three spectrum lines resulting from the 50 kc clock. Fig. 14b shows the amplitude of the final two spectrum lines obtainable with the test setup and the noise level as a function of input voltage.

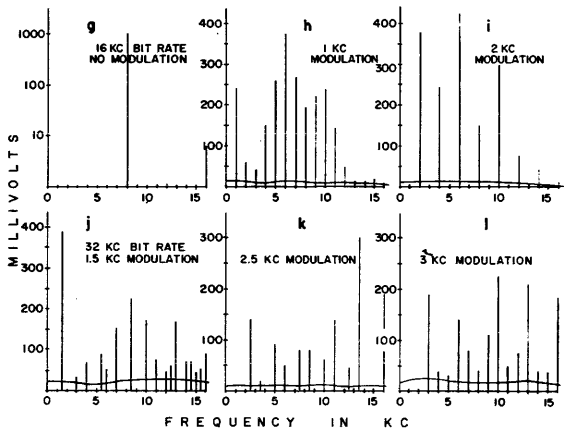


Figure 13. Spectrum Study of Flip-flop Output.

It was noted that spectra containing exclusively harmonics of the fundamental occur only when the bit rate is an integral multiple of the modulating frequency. The converse of this is not always true, however, as can be seen in Fig. 13b.

Spectrum analysis was also performed for the 2 mc bit rate and 20 kc filter cutoff up to 16 kc. Under these conditions only harmonic frequencies of the fundamental were present. Typical spectra are shown in Figs. 13d, e, and f. For bit rates several orders of magnitude higher than the modulating frequency, the question of even division depends on the fourth and higher digits in the clock frequency, and these are subject to random drift. Under these conditions only harmonics of the fundamental were observed. These amplitudes were much less than that of the fundamental.

Spectrums for low bit-rates are shown in Figs. 13g through l. This allows the spectrum analyzer to cover more of the spectrum and gives a better insight into system operation. Figs. 13g, h, and i are for a 16 kc bit-rate in which the entire spectrum may be studied. In Fig. 13g there is no input and the main component is at one half the clock frequency as has been previously noted in Fig. 2. There is only

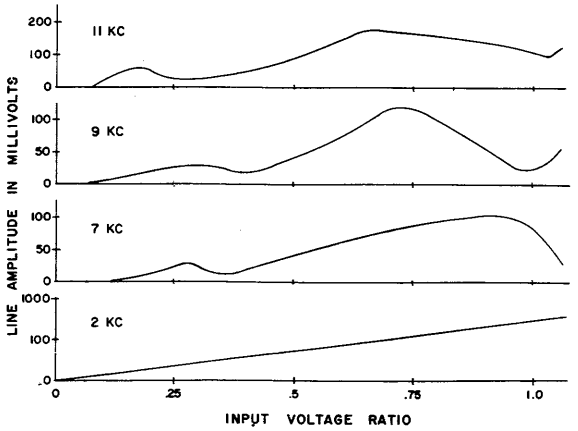


Figure 14a. Spectral Line Amplitudes Versus M.

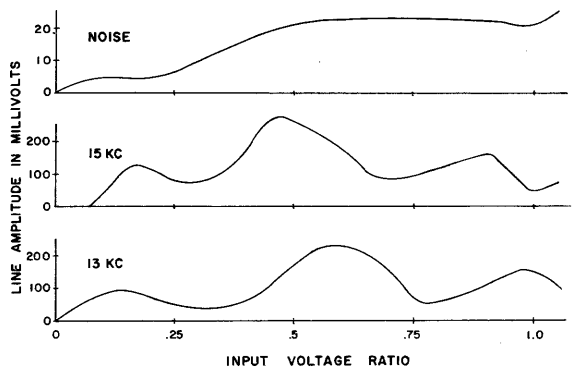


Figure 14b.

a small clock component and no measurable noise (note logarithmic millivolt scale). In Figs. 13h and i the fundamental frequency evenly divides the clock frequency and all signal harmonic frequencies are present. Note that the amplitude of some harmonics are greater than the fundamental.

A bit-rate of 32 kc is used for Figs. 13j, k, and l allowing half of the spectrum to be observed. In this case nonfractional modulating frequencies are shown. Fig. 13j has many of the harmonic frequencies missing and shows no pattern of recognizable order as compared with the two spectrums to be described below. Fig. 13k has all signal harmonics present plus additional frequencies four tenths of the way in between successive harmonics. Fig. 13-l also has all harmonics present with the intervening space equally divided by two more spectral lines. Figs. 13k and l also show some spectrum lines with amplitudes above that of the fundamental.

CONCLUSION

Providing time delay and memory in an analog computer opens up new classes of problems which need to be solved. Simulations involving transport lags, correlation studies and communications problems require the formation of time delays. Iterative techniques for solving integral equations require analog function storage.

The use of a magnetostrictive delay line for the storage medium is attractive because of the success this technique has had in digital computers. However, information must be introduced into this line in a digital manner. Although PCM is the most efficient method of digitizing, its equipment complexity and high cost make it unattractive to small computer installations. Delta sigma modulation, which encodes an analog signal into a digital code, is an inexpensive technique which has excellent linearity and good noise characteristics. Using this method of modulation in conjunction with a magnetostrictive delay line, 5 to 10 milliseconds delays may be obtained at 2 mc bit rates. The device will accommodate analog signals of up to 8 kc with phase shift below 2 degrees. The total dynamic error is within .2 per cent of half scale up to 1 kc.

Magnetostrictive delay lines are limited at present to pulse rates of 5 mc. However, advances which will be made in high speed digital storage techniques can be utilized for analog storage since the analog signal is encoded into digital form. This system, due to its fixed spacing of narrow pulses, also offers the possibility of multiplexing. That is, one delay line with its read-write circuitry could be used to store several analog functions. This would, however, bring back the problem of the use of precise timing circuits in order to separate the pulses belonging to each analog function.

APPENDIX

Delta Sigma Modulator System Equations

Consider the basic delta sigma modulator as shown in Fig. 1. The integrator output, $\epsilon(t)$ is given by

$$\frac{d}{dt}\epsilon(t) = S(t) - F(t) \qquad \text{(A-1)}$$

where $S(t)$ is the input signal and $-F(t)$ is the flip-flop output. Consider Equation A-1 evaluated at $t = nT$ where T denotes the clock period and n an arbitrary integer.

$$\frac{d}{dt}\epsilon(nT) = S(nT) - F(nT) \qquad \text{(A-2)}$$

$\frac{dt}{dt}\epsilon(nT)$ may be approximated by

$$\frac{d[(n+1)T] - \epsilon(nT)}{T}$$

hence,

$$\frac{\epsilon[(n+1)T] - \epsilon(nT)}{T} = S(nT) - F(nT) \qquad \text{(A-3)}$$

Now

$$F(nT) = \begin{cases} +E_0 \text{ if } \epsilon(nT) \geqq 0 \\ -E_0 \text{ if } \epsilon(nT) < 0 \end{cases}$$

so

$$F(nT) = E_0 \frac{\epsilon(nT)}{|\epsilon(nT)|}$$

provided that

$$\frac{\epsilon(nT)}{|\epsilon(nT)|} = 1 \text{ for } \epsilon(nT) = 0$$

Rewriting Equation A-3, we find

$$\epsilon[(n+1)T] - \epsilon(nT) = T\,S(nT) - E_0 \frac{\epsilon(nT)T}{|\epsilon(nT)|} \qquad \text{(A-5)}$$

$$\epsilon(n+1)T + \epsilon(nT)\left[\frac{T E_0}{|\epsilon(nT)|} - 1\right] = T\,S(nT) \qquad \text{(A-6)}$$

A-6 is the basic equation describing the behavior of the modified delta modulation system. Unfortunately, the presence of the absolute value term (which has as its counterpart the comparator in the actual system) does not allow an analysis of the modulator on a linear system basis. However, Equation A-6 may be solved for specific inputs either on a digital computer,[14] or by a step by step solution.

BIBLIOGRAPHY

1. KORN, G. A., New High-Speed Analog and Analog-Digital Computing Techniques: The ASTRAC System, *Electronic Industries*, July 1962.

2. ECKES, H. R., and G. A. KORN, Digital Program Control for Iterative Differential Analyzers, *ACL Memo No. 86,* University of Arizona.
3. KORN, G. A., High-Speed Iterative Analog-Digital Computation: The ASTRAC II System, *ACL Memo No. 93,* University of Arizona.
4. KORN, G. A., Analog/Hybrid Storage and Pulse Modulation, *IEEE Transactions, on Electronic Computers,* September 1963.
5. KENNEDY, J. D., Representation of Time Delays in Huskey, H. D., and G. A. Korn, *Computer Handbook,* McGraw Hill, 1962.
6. JURY, S. H., Memory and Function Generation in Analog Computers, *Military Systems Design,* January 1962.
7. DUNDON, T., Magnetostrictive Delay Lines for Digital Applications, *Computer Design,* January 1963.
8. *Hybrid Computer Course Notes,* Electronics Associates, Inc., Princeton Computation Center, Princeton, New Jersey.
9. DE JAGER, Delta Modulation, A Method of PCM Transmission Using the One Unit Code, *Phillips Res. Repts.,* Vol. 7, 1952.
10. INOSE, H. Y., YASUDA and J. MURAKAMI, A Telemetering System by Code Modulation, *IRE Transactions on Space Electronics and Telemetry,* September 1962.
11. STORER, J. E., *Passive Network Synthesis,* McGraw Hill, 1957.
12. HANDLER, H., Measurement of Phase Shift, *IEEE Transactions on Electronic Computers,* June 1963.
13. INOSE, H., and Y. YASUDA, A Unity Bit Coding Method by Negative Feedback, *IEEE Special International Issue,* November 1963.
14. TRIPP, J. S., A Deterministic Study of Delta Modulation, *Kansas State University Bulletin,* Special Report Number 17.

ACKNOWLEDGEMENT

The project described in this report is part of a hybrid analog-digital computer study directed by Professor G. A. Korn. The writers are very grateful to the Office of Aerospace Research, Information Research Division, Air Force Office of Scientific Research and to the Office of Space Sciences, National Aeronautics and Space Administration for their continuing support of this study under joint grant AF-AFOSR-89-63; and to Professors L. Matsch, Dean of Engineering and Harry Stewart, Head, Electrical Engineering Department, for their encouragement and contribution of University facilities.

A COMPUTER-SIMULATED ON-LINE EXPERIMENT IN LEARNING CONTROL SYSTEMS

J. D. Hill, G. J. McMurtry, and K. S. Fu
Control and Information Systems Laboratory
School of Electrical Engineering
Purdue University
Lafayette, Indiana

INTRODUCTION

The structure of a learning control system combines digital memory and logic circuitry with an adaptive system. Data obtained through adaptation is stored (remembered) and later utilized to improve the system performance. The learning control system will operate satisfactorily under changing environmental conditions in which an adaptive system fails to improve the performance. The general learning system operation is outlined in this paper and a simple experimental system which illustrates the improved performance is discussed. The example system is a second order plant in which the damping ratio and undamped natural frequency are considered to be affected by the environment in a piecewise constant manner. The system was simulated on a GEDA analog computer and the memory and logic functions were supplied by an IBM 1620 through IBM 1711 and 1712 analog to digital and digital to analog equipment.

LITURATURE SURVEY

The words "learning system" and "self-organizing" appear frequently in the literature in the area of automata studies and artificial intelligence. Hawkins[1] has given an excellent review on self-organizing systems as well as an extensive bibliography, while Minsky[2] has reviewed various aspects of artificial intelligence including hill-climbing, methods of classification for pattern recognition, reinforcement learning, problem solving machines and various other aspects of artificial intelligence. Nearly all of these systems utilize some form of memory and do improve their performance with time. However, they require an off-line training period involving a human operator to judge whether or not their response to a given stimulus is satisfactory, i.e., they lack a built-in index of performance.

A typical example of this is given by Gabor, Wilby and Woodcock[3] who have constructed a universal nonlinear filter which will organize its internal parameters in such a way as to act as an optimum filter, predictor, or simulator of an unknown mechanism, depending upon how it is trained. The important point is, however, that it must be trained by a human operator before it can function in the desired manner. This is typical of the general class of so-called learning or self-organizing pattern recognition machines. "Pandemonium,"[4] "Perceptron,"[5] and "Adaline"[6] are other pattern recognition devices with self-organizing capabilities. All, however, require off-line training by human operators.

Thus, though some of the techniques which have been developed for pattern recognition and automata theory may be useful when applied to the learning control problem, there has as yet

been very little direct application to control systems.

In the hierarchy of automatic control systems, adaptive systems are more sophisticated than standard control systems. Truxal[7] gives a good general review of adaptive control systems while Mishkin and Braun[8] discuss several specific examples. The next advancement beyond adaptive systems appears to be in the area of learning control systems.

Krug and Letskii[9] have suggested the use of a learning process for optimum control. In their paper they suggest that the optimum control of slow but complex processes, such as chemical processes, might be found by a systematic evaluation of input and output data and an index of performance. Probabilistic methods are suggested for organizing the memory for optimum search procedure. The system as outlined is very general and a human operator is left to make the final decision as to whether the control found by the so-called automaton is satisfactory. Only cursory attention is given to the possibility of replacing the human operator.

A considerable effort has been directed toward the investigation of learning in automatic control systems at the Control and Information Systems Laboratory, School of Electrical Engineering, Purdue University. An informal introduction to learning control systems has been given by Fu.[10] One of the specific systems investigated at Purdue University[11] is discussed in this paper.

Philosophy of Learning Control Systems

The structure of a learning control system, as visualized at present, is distinguished from an adaptive system mainly in that in addition to a conventional adaptive system the learning system has memory and logic. Data obtained through adaptation is stored and later utilized to improve the system performance. Thus, whereas an adaptive system may take advantage only of its immediate past experience, a learning system is able to recall and use plant adjustment data obtained through adaptation while operating under similar environmental conditions in past time. An adaptive system will optimize a slowly time varying plant to a given index of performance (IP), often through a hill-climbing technique, by modifying the controller or plant parameters. Thus, if a sufficient range of parameters is available, the adaptive system will become optimum for a given index of performance. A basic constraint is that the plant should vary slowly enough that the adaptive loop is able to track the minimum of the index of performance and maintain relatively constant performance.

In some instances, the plant parameters vary so quickly that the adaptive loop cannot maintain optimum performance although some adaptive action does take place. This is one type of situation in which the learning system proposed by the authors may be applicable. Learning systems of the type discussed also yield superior performance in systems with slowly varying parameters which change suddenly but are relatively constant between changes. The learning system would be designed so that it would store in memory a quantized measure of the plant parameters which are varying or of the environment causing the variation, the best index of performance obtained by the adaptive system, and the corresponding corrections necessary to obtain this performance.

It is important to realize that the hill-climbing technique used in the adaptive portion of the system may not converge to the minimum of the IP surface if the measurements of the IP are noisy. Thus all environmental parameters affecting the IP must be measured. Only if a separate hill-climb is performed for each combination of environmental parameters and all environmental parameters are considered will the IP surface be free of noise and the adaptive portion of the system operate properly.

Thus when a previously occurring set of plant parameters is again encountered, the best learned corrections would be immediately set from memory and adaptation carried out from that point. The system would, of course, be designed so that the contents of the memory are continuously updated by storing the most recent best settings of the corrections.

Assuming then, that the varying parameters may be quickly identified, and that the memory interrogation is faster than the adaptive action, the system performance, after sufficient operat-

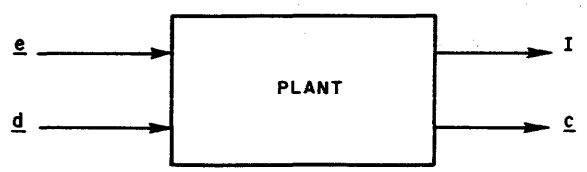


Figure 1. General Plant.

ing time, would approach the optimum performance.

In the general learning situation the plant and plant parameters are shown in Fig. 1. In Fig. 1, $e = (e_1, e_2, \ldots, e_n)$ is the environment vector which is considered to be measurable but uncontrolled. It includes all plant inputs and environmental factors which cannot be changed in order to improve the IP. $d = (d_1, d_2, \ldots, d_p)$ is the plant parameter adjustment vector and includes plant inputs which may be changed in order to improve the IP. c = plant output vector, and I is the plant index of performance.

Now each component e_i of the plant environment vector e, corresponds to a separate environmental or input variable such as temperature, pressure, etc.

For measurement purposes, each of these variables, e_i, is assumed to be quantized into q_i levels where the q_i's for different values of i ($i = 1, \ldots, n$) are not necessarily equal. Then the number N of possible different environment vectors e is

$$N = \prod_{i=1}^{n} q_i$$

e will thus be relabeled $e = e^r$; $r = 1, 2, \ldots, N$ when referring to the r^{th} possible environment.

The index of performance, I, is considered to be a scalar function of e^r, d and c, i.e.,

$$I = f(e^r, d, c)$$

where $c = F(e^r, d)$. Thus, for a given e^r, I has at most a single minimum as a function of d. I is then a p-dimensional hyper-surface in the adjustment parameters d_j ($j = 1, 2, \ldots, p$). There results, then, a p-dimensional hill-climbing problem corresponding to each different e^r.

There are several methods of hill-climbing the d vector to the minimum of the IP surface, but the one chosen here is to hill-climb each component of d separately and sequentially, the best values of each of the p parameters being stored in memory corresponding to each e^r, along with the corresponding direction of increase or decrease in each parameter of d and the lowest value of I. The stored adjustment vector corresponding to e^r is labeled d^{rM}. (Superscript M indicates a stored or memorized parameter.)

The operation of the system is then as follows. Upon recognition of the occurrence of e^r, d is set to d^{rM} from a search of memory, and hill-climbing proceeds. The computer program should be arranged to facilitate a fast search of memory for previously occurring e^r. This may be implemented by programming the computer to search the most frequently occurring e^r's first. The program must also be capable of setting up new storage locations for e^r's which have not previously occurred.

Now let us consider that a particular e^r occurs, given that it has previously occurred k times. Immediately $d^{rM}(k)$ is set from memory and parameter d_j is to be adjusted, where:

$$j = k - a\,p$$

$$p \geqq j > 0$$

$$a = \begin{cases} 0 & 0 < k \leqq p \\ 1 & p + 1 \leqq k \leqq 2p \\ \text{etc.} & \end{cases}$$

These conditions are applied to indicate sequential nature of adjustment of d_j's as mentioned previously.

Then $d_j^r(k+1)$ is adjusted to

$$d_j^r(k+1) = d_j^{rM}(k) + s_j^{rM}(k)\ \Delta d_j$$

where

Δd_j = increment of d_j

$s_j^{rM}(k) = \pm 1$ (stored direction in which to increment $d_j^{rM}(k)$)

Let us now consider the logic required to program the hill-climbing operation and the modification of computer memory. Define

$$m_j^r(k+1) = \begin{cases} 1 \text{ for } (I^r(k+1) - I^{rM}(k)) \leqq 0 \\ 0 \text{ for } (I^r(k+1) - I^{rM}(k)) > 0 \end{cases}$$

where

$I^r(k+1) = I(e^r, d^r(k+1))$

$I^{rM}(k) = I(e^r, d^{rM}(k))$

The memorized compensation parameter then becomes

$$d_j^{rM}(k+1) = d_j^{rM}(k) + m_j^r(k+1)\, s_j^{rM}(k)\, \Delta d_j$$

The memorized compensation vector is given by

$$d^{rM}(k+1) = d^{rM}(k) - d_j^{rM}(k) + d_j^{rM}(k+1)$$

where

$$d_j^{rM}(k) = (0_1, 0_2, \ldots 0_{j-1}, d_j^{rM}(k), 0_{j+1}, \ldots 0_p)$$

$$d_j^{rM}(k+1) = (0_1, 0_2, \ldots 0_{j-1}, d_j^{rM}(k+1), 0_{j+1}, \ldots 0_p)$$

The memorized value of the index of performance becomes

$$I^{rM}(k+1) = I^{rM}(k) + m_j^r(k+1)\,(I^r(k+1) - I^{rM}(k))$$

and the stored direction in which to increment $d_j^{rM}(k+1)$ is given by

$$s_j^{rM}(k+1) = -\operatorname{sgn}\,[(I^r(k+1) - I^{rM}(k))\,(d_j^r(k+1) - d_j^{rM}(k))]$$

In the above development, all environmental parameters affecting the system are assumed to be measurable and measurement noise is neglected. All environmental parameters e^r are assumed to be constant during one hill-climbing step. In the experimental example the same assumptions were made except that e^r was assumed constant for four hill-climbing steps.

Experimental System

A learning system of the type discussed above with second order plant has been simulated on the IBM 1710-GEDA hybrid computer installation. The system was assumed to be subjected to a pseudo-random sequence of environmental conditions which change the damping coefficient and undamped natural frequency of the system at discrete intervals. The environment was changed sufficiently often that purely adaptive action (i.e, hill-climbing on the IP surface) could not optimize the system during the period of constant environment.

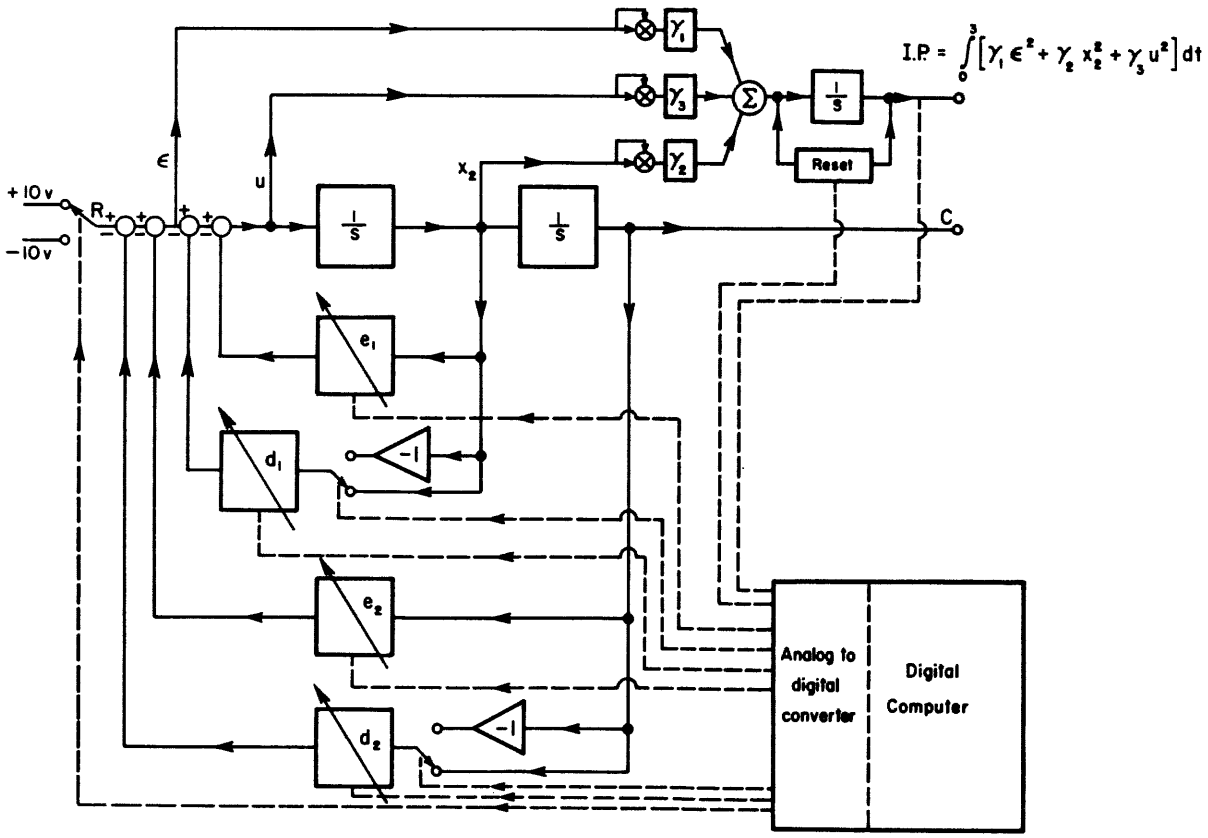


Figure 2. Experimental Learning System.

The system was subjected to a fixed amplitude square wave input in order to facilitate the computation of an index of performance of the quadratic type. In the plant, as shown in Fig. 2, e_1 and e_2 are environmental parameters which are constrained to remain constant over two periods of the square wave input. Thus, at each occurrence of an (e_1, e_2) pair four computations of the IP are carried out, one at each transition of the square wave. After each computation of the IP, the compensation parameters d_1 and d_2 are adjusted by a two-dimensional hill-climbing technique described above in order to minimize the performance index. Memory is used to store the current best values of d_1 and d_2 for a given set of (e_1, e_2) values, as well as the directions in which d_1 and d_2 were being adjusted, and the best value of the IP found previously. Thus when a previously occurring (e_1, e_2) pair reoccurs, the best values of d_1 and d_2 are set from memory and the best direction to increment d_1 and d_2 is known. Adaptation then proceeds and the latest values of the best d_1 and d_2, direction and IP replace the old values in memory.

For the particular system under investigation, e_1 and e_2 were each allowed to take on five different values, i.e., $q_1 = q_2 = 5$. Thus twenty-five different (e_1, e_2) combinations were possible. In order to simulate the limitations on computer memory capacity, only the sixteen most probable (e_1, e_2) combinations were allowed to have corresponding d_1, d_2, etc. data stored with them. In this system no compensation or hill-climbing took place on the nine least probable (e_1, e_2) combinations. Probabilities were computed by counting occurrences of (e_1, e_2) pairs and a continuous check was made for the 16 most probable. In the hill-climbing technique, the increments used for changing d_1 and d_2 were of fixed size and were not reduced even after learning was essentially complete.

The changes in e_1 and e_2 were generated by storing in memory a pseudo-random sequence of (e_1, e_2) pairs two hundred long. The (e_1, e_2) pairs were set from memory on the digital pots just prior to the square wave transition and the sequence of (e_1, e_2) pairs repeated after each two hundred values (fifty different (e_1, e_2) pairs since each pair was repeated at least four times in a row).

The performance index (IP) used was of the form

$$IP = I = \int_0^T (\gamma_1 \epsilon^2 + \gamma_2 x_2^2 + \gamma_3 u^2)\, dt$$

where ϵ, x_2, u are as defined in Fig. 2, and γ_1, γ_2, γ_3 may be changed as desired. In this example, the parameter e_1 was allowed to take on the values 4, 5, 6, 7, or 8 and e_2 to have the values 20, 30, 40, 50 or 60.

An outline of the digital computer program and flow diagram for the system under consideration are given in Appendix A.

Discussion of Results

An investigation of the IP surface, for the particular system and performance indices under investigation, was carried out. The system may be redrawn as shown in Fig. 3 if we let $a = e_1 + d_1$ and $b = e_2 + d_2$. Let the input R(s) be a step input of amplitude A. Then

$$R(s) = \frac{A}{s}$$

$$\epsilon(s) = \frac{A(s + a)}{s^2 + as + b}$$

$$x_2(s) = \frac{A}{s^2 + as + b}$$

$$u(s) = \frac{As}{s^2 + as + b}$$

For an index of performance given by

$$I = \int_0^\infty (\gamma_1 \epsilon^2 + \gamma_2 x_2^2 + \gamma_3 u^2)\, dt$$

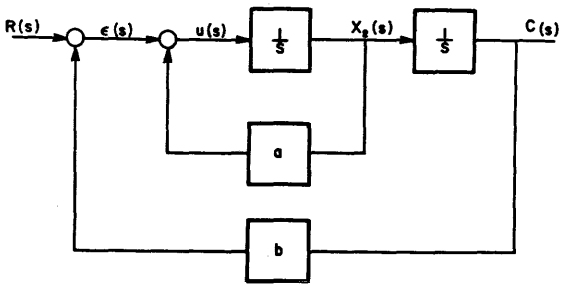


Figure 3. Simplified System Block Diagram.

Parseval's theorem yields

$$I = \frac{1}{2\pi_j} \int_{-j\infty}^{j\infty} \{\gamma_1[\epsilon(s)\ \epsilon(-s)] + \gamma_2[x_2(s)\ x_2(-s)] + \gamma_3[u(s)\ u(-s)]\}\, ds$$

After some manipulation, the integrand becomes

$$\frac{A^2(\gamma_1 + \gamma_3)\left[s + \sqrt{\frac{\gamma_2 + a^2\gamma_1}{\gamma_1 + \gamma_3}}\right]\left[-s + \sqrt{\frac{\gamma_2 + a^2\gamma_1}{\gamma_1 + \gamma_3}}\right]}{(s^2 + as + b)(s^2 - as + b)}$$

The IP may be evaluated from standard tables[12] to yield

$$I = \frac{A^2[b(\gamma_1 + \gamma_3) + \gamma_2 + a^2\gamma_1]}{2ab}$$

Then for given γ_1, γ_2, and γ_3, contours of IP may be plotted in the (a, b) plane.

For the performance index used in this example

$$\gamma_1 = 0, \quad \gamma_2 = 1, \quad \gamma_3 = 1$$

Thus

$$I = \frac{A^2[b + 1]}{2ab} = \frac{k[b + 1]}{ab}$$

The constant IP contours for this performance index are shown in Fig. 4 for k = 1.0. We note that for this case, the index of performance is very insensitive to $e_2 + d_1 = b$ but very sensitive to $e_1 + d_1 = a$. This theoretical conclusion was verified experimentally as may be seen from the trajectories plotted in Fig. 4 for several (e_1, e_2) values. The system trajectories required approximately two and one-half hours operation to reach the points indicated. The hill-climbing was not completed at this stage but the time was sufficient to illustrate that the system was learning. In this

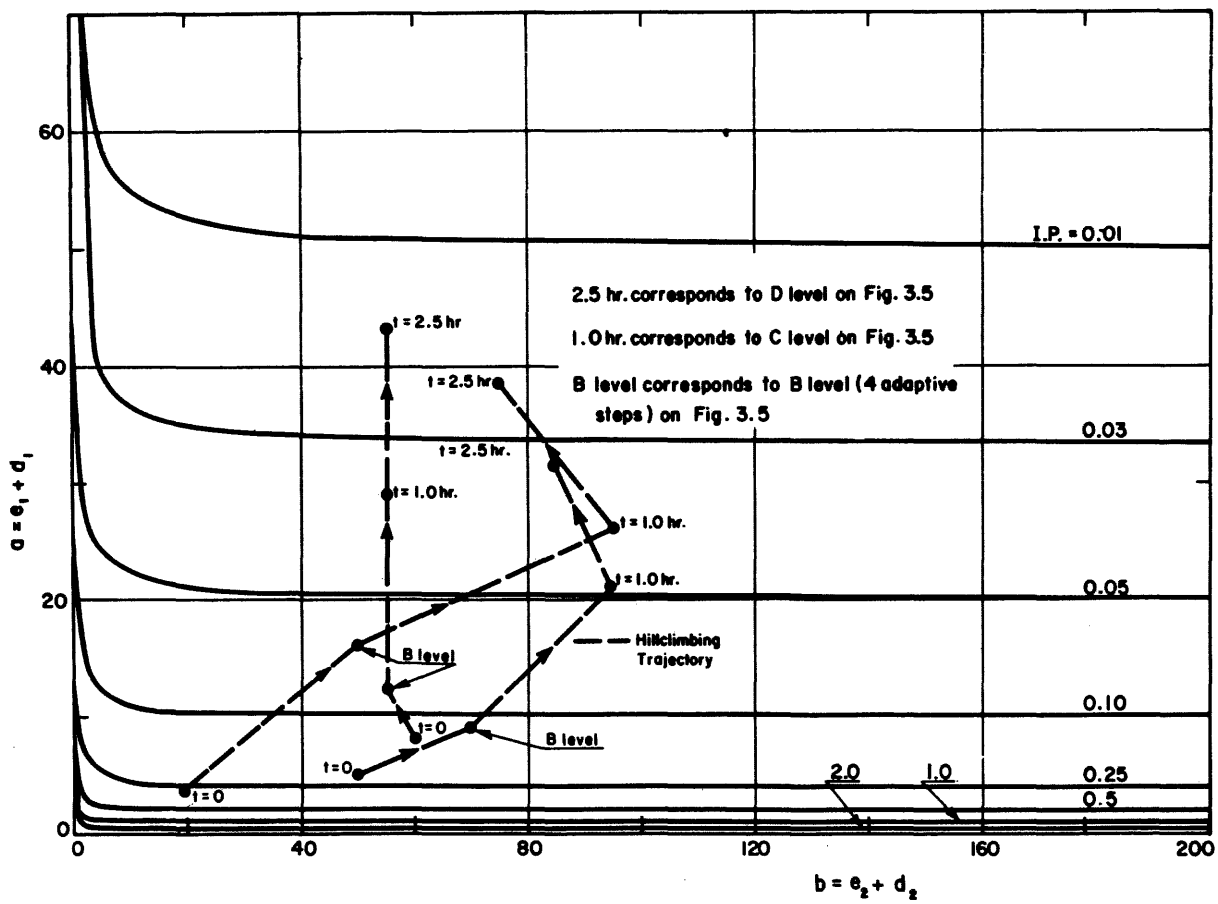


Figure 4. Index of Performance for $I = \frac{k(b + 1)}{ab}$

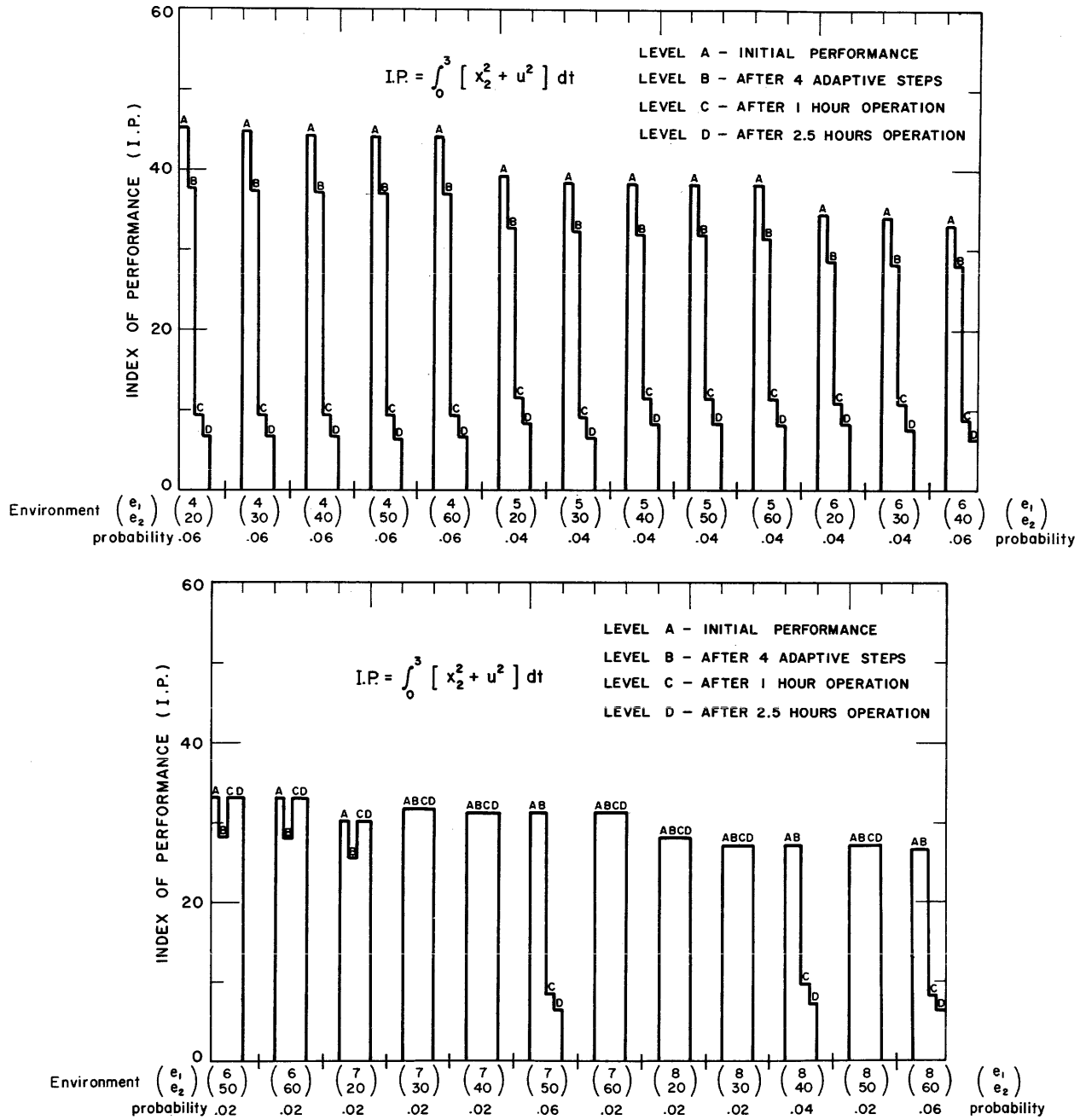


Figure 5. Learning System Performance.

length of time the IP decreased by a factor of approximately six. The improvement in IP for each different (e_1, e_2) pair may also be seen from Fig. 5, which was plotted from computer type-out data as discussed in Appendix A. Note the IP scale in Fig. 5 was not normalized as are the contours in Fig. 4. It is noted that this particular performance index becomes zero as $a \to \infty$, $b \to \infty$, and does not have a particular minimum. Thus after sufficient operating time, the d_1 and d_2 adjustments would simply reach the maximum values allowed for in the system design (100 and 1000 respectively). It appears that the IP is zero for $b = -1$ but this value

of b invalidates the derivation for the IP since Parseval's theorem is valid only for asymptotically stable systems.

Fig. 5 is a plot of IP versus (e_1, e_2) pairs for the index of performance investigated. Four points are plotted for each (e_1, e_2) pair corresponding to four different times in the learning process. Level A represents the IP measured before compensation. The particular sequence of (e_1, e_2) pairs given in this test was such that the first twenty-five pairs seen by the system were all different (from left to right as plotted in Fig. 5). After all twenty-five pairs occurred once, the order of occurrence was no longer the same. Level B represents the IP for each pair after the first four adaptive steps for each (e_1, e_2). The first sixteen (e_1, e_2) pairs seen by the system were compensated initially and information was stored. The last nine, in order of first appearance, had no information initially stored, and thus showed no improvement.

Level C represents the IP after approximately one hour of running. Here it is seen that the system had determined the sixteen (e_1, e_2) pairs occurring most frequently (most probable) and had continuously learned on them. The nine least probable pairs had not improved from their initial values.

Level D represents the IP at the conclusion of the test for each pair. Here again, of the sixteen most probable, the ones with highest probability learned faster. Of course adaptation would normally be carried out on all (e_1, e_2) pairs and not just on the 16 most probable.

The learning system then operates as an adaptive system when a given (e_1, e_2) pair or environment first occurs, but has the capability of utilizing past information about the best compensation parameter setting when a previously occurring environment reoccurs. In this case, given sufficient operating time, the learning system is capable of reducing the index of performance to the minimum possible value for the 16 most probable (e_1, e_2) pairs (environmental conditions) even though the environment is changing so rapidly that an ordinary adaptive system would fail to improve the index of performance significantly.

Concluding Remarks

Several particular problems currently under investigation are the application of stochastic approximation techniques[13,14,15] to hill-climbing when the IP surface is disturbed by noise due to unmeasurable environmental parameters and the application of pattern recognition techniques to measurements of the IP surface in order to increase the rate of learning. Present studies are also concerned with methods of quantizing the environment parameters to yield efficient learning operation subject to the constraint of finite computer memory capacity.

The principle of learning outlined in this paper, that is, partial hill-climbing of the IP surface at each occurrence of a particular environment and memorization of the best compensation found for that environment, seems to be a relatively simple idea; and yet, as illustrated by the example, it yields great improvement in the system performance over that for a simple adaptive system. It is felt that this concept of learning control is particularly applicable to process control.

Acknowledgement

The authors wish to thank Dr. J. E. Gibson for his stimulating discussions. This work is in part supported by the Research Contract AF AFOSR 62-351 and National Science Foundation Grant G-14609.

APPENDIX

Description of the Computer Program

The flow diagrams shown in Figs. 6, 7, and 8 illustrate the system logic and timing. A more detailed examination follows:

1. The (e_1, e_2) values were punched on cards and stored in memory at the beginning of the program. The program did not take advantage of this fact in determining the probability of each (e_1, e_2) pair.

2. The type-out of results was controlled manually by the computer operator by means of a Program Switch. If the switch was on and a cycle of 200 (e_1, e_2) values had been completed, the type-out was performed, except at the beginning of the program where type-out was performed after the first 100 (e_1, e_2) values.

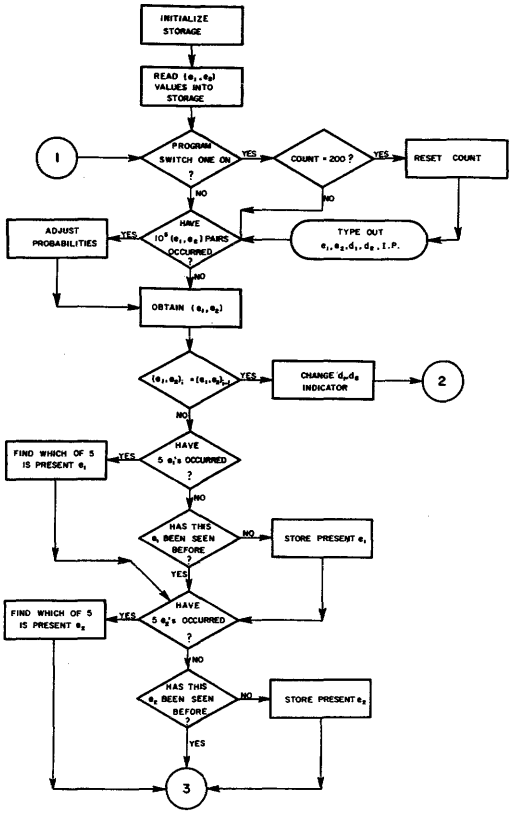


Figure 6. Flow Diagram (Part I).

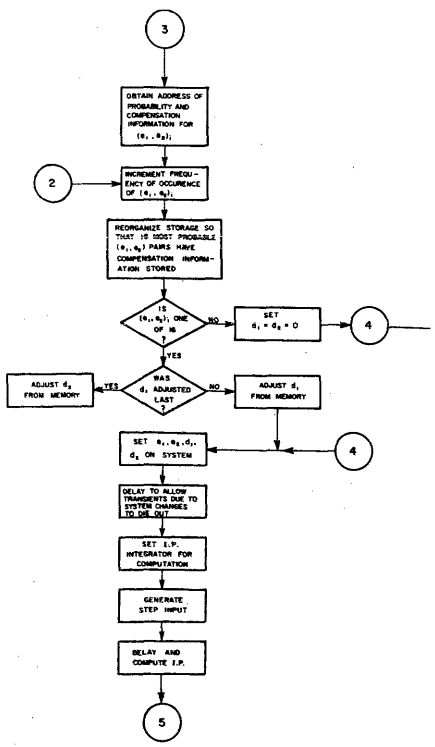


Figure 7. Flow Diagram (Part II).

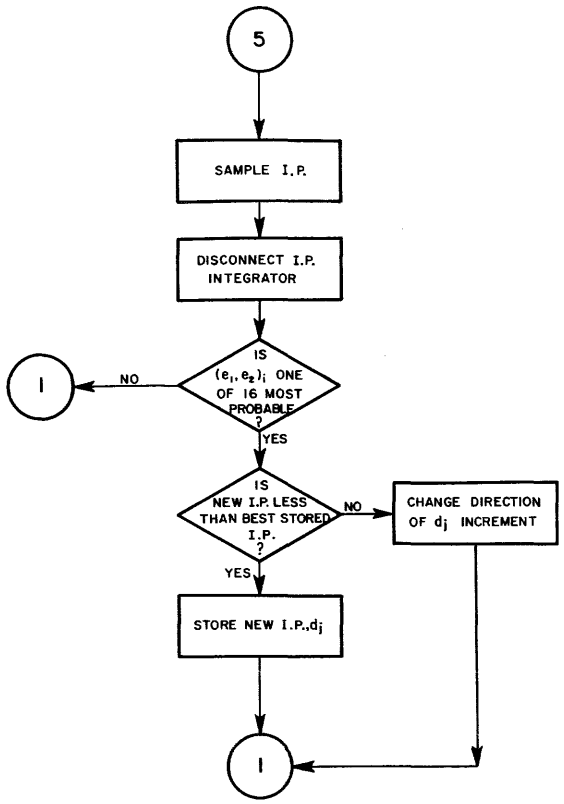


Figure 8. Flow Diagram (Part III).

3. When 10^5 (e_1, e_2) values had occurred, the total number of recorded occurrences of each (e_1, e_2) pair was divided by ten and a counter was reset to count the next 10^5 values. Then each new occurrence of a particular (e_1, e_2) pair caused the number of recorded occurrences for that (e_1, e_2) to be increased by one. Thus the more recent occurrences were weighted more heavily.

4. When a new pair, (e_1, e_2), was selected, it was compared to the previous pair, $(e_1, e_2)_{i-1}$. If they were the same, the search procedure for (e_1, e_2) and its information was bypassed, since the location of information for $(e_1, e_2)_i$ was the same as for $(e_1, e_2)_{i-1}$. As shown in 8 below, an indicator was placed in memory for each (e_1, e_2) pair showing which d_j was operated on last. Then the program would operate on the other d_j when that (e_1, e_2) pair occurred again. In order for the program to operate on the same d_j when an (e_1, e_2) pair occurred twice or more in a row, the indicator was changed when $(e_1, e_2)_i$ was determined to be the same as $(e_1, e_2)_{i-1}$.

5. If $(e_1, e_2)_i \neq (e_1, e_2)_{i-1}$, then memory was searched to see if e_1 had ever occurred previously. Next e_2 was checked for previous ocurrence. Then $(e_1, e_2)_i$ was assigned an address where information was stored concerning its probability and location of index of performance and compensation information. The probability was incremented next (actually the number of recorded occurrences for $(e_1, e_2)_i$ was increased by one).

6. At this point memory was organized so that only the sixteen most frequent (e_1, e_2) pairs had compensation information stored. If $(e_1, e_2)_i$ had no information stored, then d_1 and d_2 were set to zero.

7. The following information was stored for the sixteen most probable (e_1, e_2) pairs:

a) whether d_1 or d_2 was operated on most recently

b) the value of d_j $(j = 1, 2)$ that resulted in the best (lowest) IP value

c) the slope (direction of change) for d_j $(j = 1, 2)$

d) the sign of d_j $(j = 1, 2)$

e) the best previous value of IP.

8. If information was stored for $(e_1, e_2)_i$, the program next determined from an indicator which d_j was operated on last. Assume this was d_1. The stored value of d_1 was then set, and operations commenced on d_2. The indicator was then set to show that the program acted on d_2 last (see 4 above). Next, the slope was determined.

a) If the slope was positive, the program added the standard increment, $\triangle d$, to d_2 and the sum was compared with 9999, since this was the maximum setting on the digital potentiometer. If the sum was greater than 9999, then d_2 was set to 9999. If not d_2 was set equal to the sum in question.

b) If the slope was negative, and d_2 was greater than $\triangle d$, then $\triangle d$ was subtracted from d_2. If d_2 was less than $\triangle d$, the sign of d_2 was changed and the magnitude of d_2 was not changed. The sign of d_j was made negative or positive, respectively, by switching an amplifier in or out of the feedback loop (see Fig. 2). If the sign was negative the amplifier was in, thus yielding a subtractive correction.

c) Note that the previous values of d_1 and d_2 were left in memory. The new d_j were stored later only if the new IP to be measured was lower than the previously stored value of the IP (see 13 below).

9. The proper values of e_1, e_2, d_1 and d_2 were then set on the analog computer and the IP integrator was set for integration.

10. The step input was generated next, using a comparator on the GEDA. A delay followed, during which the IP was computed. At the end of the delay, the IP was measured and the IP integrator was disconnected and reset to zero.

11. If no information was stored for $(e_1, e_2)_i$, the system had completed the cycle and the next (e_1, e_2) pair was obtained.

12. When information was stored for $(e_1, e_2)_i$, and if the measured IP was greater than the stored best previous IP, then the slope indicator was changed on d_j, so that d_j would be incremented in the opposite direction the next time the same (e_1, e_2) pair occurred. If the slope of d_j was changed two consecutive times, then the present measured value of IP was stored and d_1 and d_2 were not changed in memory. This procedure was implemented to allow for the possibility that an erroneously low value of IP was stored due to any random noise pulses introduced into the system at some previous time. If the slope of d_j had not changed twice consecutively, only the slope indicator was changed in memory and the system had completed the cycle.

13. If the measured value of IP was lower than the stored value, then the measured value, along with the present d_1 and d_2 values, were placed in memory and the cycle was complete.

References

1. J. K. HAWKINS, "Self-Organizing Systems —A Review and Commentary," Proc. of the IRE, Vol. 49, No. 1, January, 1961.
2. M. MINSKY, "Steps Toward Artificial Intelligence," Proc. of the IRE, Vol. 49, No. 1, January, 1961.
3. D. GABOR, W. P. L. WILBY, and R. WOODCOCK, "A Universal Nonlinear Filter, Predictor and Simulator which Optimizes Itself by a Learning Process," IEE Proc., Vol. 108, Part B, 1961, p. 422.
4. O. G. SELFRIDGE, "Pandemonium: A Paradigm for Learning," Proceedings of the Symposium on Mechanization of Thought Processes, National Physics Laboratory, Teddington, England, Her Majesty's Stationery Office, London, Vol. I, pp. 513-531, 1959.
5. F. ROSENBLATT, "The Perceptron, A Theory of Statistical Separability in Cognitive Systems," Cornell Aeronautical Laboratory, Tr. No. VG-1196-6-1, January, 1958.
6. B. WIDROW, "Pattern Recognition and Adaptive Control Symposium and Panel Discussion on Discrete and Adaptive Processes," JACC, June, 1962.
7. J. G. TRUXAL, "Adaptive Control," Proceedings of the International Federation of Automatic Control, 1963.
8. E. MISHKIN and L. BRAUN, "Adaptive Control Systems," McGraw-Hill, 1961.
9. G. K. KRUG and E. K. LETSKII, "A Learning Automaton of the Tabular Type," Automation and Remote Control, Vol. 22, No. 10, March, 1962.
10. K. S. FU, "Learning Control Systems," COINS Symposium, June 17-18, 1963, Evanston, Illinois.
11. J. E. GIBSON, K. S. FU. et al., "Philosophy and State of the Art of Learning Control Systems," Report TR-EE63-7, CISL, Purdue University, November, 1963, AFOSR-5144.
12. H. M. JAMES, N. B. NICHOLS, and R. S. PHILLIPS, "Theory of Servomechanisms," pp. 369-370, McGraw-Hill, 1947.
13. H. ROBBINS and S. MUNRO, "A Stochastic Approximation Method," Annals of Mathematical Statistics, Vol. 22, 1951, pp. 400-407.
14. J. KIEFER and J. WOLFOWITZ, "Stochastic Estimation of the Minimum of a Regression Function," Annals of Mathematical Statistics, Vol. 23, 1952, pp. 462–466.
15. H. J. KUSHNER, "Hill-Climbing Methods for the Optimization of Multi-Parameter Noise Disturbed Systems," Trans. of ASME, Series D, Jour. of Basic Engineering, Vol. 85, No. 2, June, 1963.

A HEURISTIC PROGRAM TO SOLVE GEOMETRIC-ANALOGY PROBLEMS

Thomas G. Evans
Air Force Cambridge Research Laboratories (OAR)
Bedford, Massachusetts

INTRODUCTION

The purpose of this paper is to describe a program now in existence which is capable of solving a wide class of the so-called 'geometric-analogy' problems frequently encountered on intelligence tests. Each member of this class of problems consists of a set of labeled line drawings. The task to be performed can be concisely described by the question: 'figure A is to figure B as figure C is to which of the given answer figures?' For example, given the problem illustrated as Fig. 1, the geometric-analogy program (which we shall subsequently call ANALOGY, for brevity) selected the problem figure labeled 4 as its answer. It seems safe to say that most people would agree with ANALOGY's answer to this problem (which, incidentally, is taken from the 1942 edition of the *Psychological Test for College Freshmen* of the American Council on Education). Furthermore, if one were required to make explicit the reasoning by which he arrived at his answer, prospects are good that the results would correspond closely to the description of its 'reasoning' produced by ANALOGY.

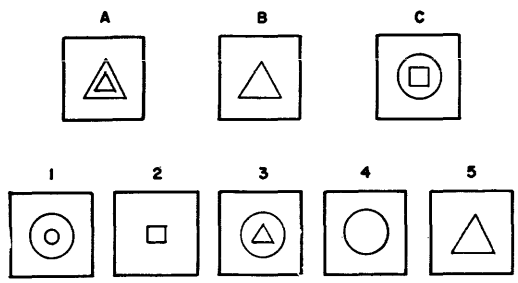


Figure 1.

At this point, a large number of questions might reasonably be asked by the reader. Four, in particular, are:

(i) Why were problems of this type chosen as subject matter?

(ii) How does ANALOGY go about solving these problems?

(iii) How competent is ANALOGY at its subject matter, especially in comparison to human performance?

(iv) What has been learned in the construction of ANALOGY and what implications might this study have for the further development of problem-solving programs in general?

The remainder of this paper constitutes an attempt to answer these questions in some detail. We first deal with a variety of motivations for this investigation and attempt to place it in the context of other work in related areas. Next we turn to detailed consideration of the problem type and of the mechanism of the ANALOGY program. Finally, we present some answers to

the remaining two questions raised above. (A more detailed discussion of all these issues can be found in Ref. 1).

Motivations and Background

In our opinion ample general justification for the development and study of large heuristic problem-solving programs has been provided (both through argument and through example) by previous workers in this area. We shall not attempt to add to it. Given that one is interested in the construction of such programs, a number of reasons can be advanced for the choice of geometric-analogy problems as a suitable subject matter. Some of these are:

(i) Problems of this type require elaborate processing of complex line drawings: in particular, they require an analysis of each picture into parts and the determination and use of various relationships among these parts. This is an interesting problem *per se* and one which can reasonably be expected to be of great practical importance in the near future.

(ii) The form of the problems requires one to find a transformation that takes figure A into figure B and takes figure C into exactly one of the answer figures. This situation provides a natural opportunity for trying out certain ideas about the use of explicit internal 'descriptions' (here, of both figures and transformations) in a problem-solving program. Furthermore, more speculatively, it presents an interesting paradigm of 'reasoning by analogy,' a capacity which may play a large role in far more sophisticated problem-solving programs in the future. (In Section 5 we discuss the possible relevance of ANALOGY to the introduction into problem-solving programs of more powerful learning mechanisms than have yet been achieved.)

(iii) Problems of this type are widely regarded as requiring a considerable degree of intelligence for their solution and in fact are used as a touchstone of intelligence in various general intelligence tests used for college admission and other purposes. This suggests a non-trivial aspect of any attempt to mechanize their solution.

We shall now attempt very briefly to place ANALOGY in the context of earlier work in related areas. Two aspects of ANALOGY must be considered:

(i) ANALOGY contains a substantial amount of machinery for the processing of representations of line drawings, including decomposition into subfigures, calculation of relations between figures, and 'pattern-matching' computations. Thus we must relate it to other work in picture processing and pattern recognition.

(ii) ANALOGY is a complex heuristic problem-solving program, containing an elaborate mechanism for finding and 'generalizing' transformation rules. Thus we must relate it to other work on the development of problem-solving programs.

We turn first to the picture-processing aspect. The essential feature of the treatment of line drawings by ANALOGY is the construction, from relatively primitive input descriptions, of more 'abstract' descriptions of the problem figures in a form suitable for input to the rule-finding program. The fundamental programming technique underlying this method is the use of a list-processing language, in this case LISP,[2,3] to represent and process the figures in question. Work in picture processing, for pattern-recognition purposes, involving some elements of description, is found in Grimsdale *et al.*,[4] Marill *et al.*,[5] and Sherman,[6] among others. Sutherland[7] and Roberts[8] have used, for quite different purposes, internal representations of line drawings similar in some respects to those used in ANALOGY. Kirsch[9] has worked with complex line drawings primarily as a vehicle for programs involving the analysis of English-language sentences pertaining to such pictures. Hodes[10] and Canaday[11] have used LISP expressions for figure description in much the same way that we have, though the development of machinery for manipulating such descriptions was, of necessity, carried much further in ANALOGY. Evidently the first advocacy of 'scene description' ideas (for use in pattern recognition) occurs in Minsky.[12]

To place ANALOGY with respect to other work with problem-solving programs, we shall simply list a number of developments in the construction of problem-solving programs which have influenced, in a general way, our approach to the design of ANALOGY. These include LT

(the Logic Theorist)[13] and, more recently, GPS (the General Problem Solver)[14] of Newell, Simon, and Shaw, the plane-geometry theorem-prover[15] of Gelernter and Rochester, and SAINT, the formal integration program of Slagle.[16]

Summary of the Solution Process, with Example

To exhibit as clearly as possible the entire process carried out by ANALOGY, we now sketch this process, then examine its operation on an example. The sample problem we shall be considering is shown as Fig. 2 (where the Pi's are not part of the problem figures but labels keying the corresponding parts of the figures to

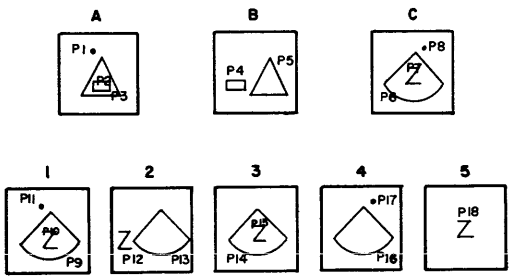


Figure 2.

expressions we shall give below). Before treating the example, we shall summarize the entire solution process. Given a problem such as that above, ANALOGY proceeds as follows: First, the input descriptions of the figures are read. Currently these descriptions, given as LISP expressions in a format to be illustrated below, are hand-made; however, they could well be mechanically generated from scanner or light-pen input by a relatively straightforward, quite 'unintelligent' program embodying line-tracing techniques already described in the literature. The descriptions represent the figures in terms of straight line segments and arcs of circles (to any desired accuracy, at the cost of longer and longer expressions). Examples of the descriptions are given below.

The first step taken by ANALOGY is to decompose each problem figure into 'objects' (subfigures). The decomposition program originally written, which was sufficient to handle many

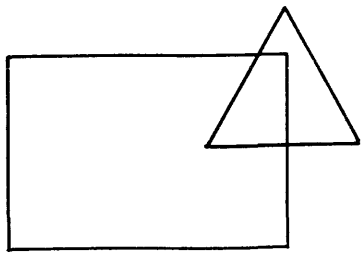


Figure 3a.

cases, including the example to be discussed below, was quite simple. It merely separated a problem figure into its connected subfigures; e.g., figure A of the above example consists of the three objects labeled P1, P2, and P3. It later became desirable to have a more sophisti-

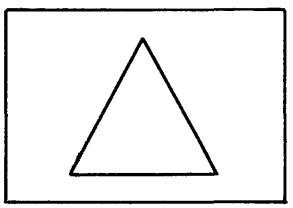


Figure 3b.

cated decomposition program with, in particular, the capability of separating overlapped objects on appropriate cues. For example, suppose problem figure A is as in Fig. 3a and figure B is as in Fig. 3b. The decomposition program should be able to separate the single object of figure A into the triangle and rectangle on the basis that they appear in figure B, from which point the remaining mechanism of parts I and II could proceed with the problem. While a decomposition program of the full generality desirable has not yet been constructed, the most recent version of the program is capable, in particular, of finding all occurrences of an arbitrary simple closed figure x in an arbitrary connected figure y; for each such occurrence the program can, if required, separate y into two objects: that occurrence of x and the rest of y (described in the standard figure format—note that this 'editing' can be rather complex: connected figures can be split into non-connected parts, etc.).

The type of decomposition illustrated above might be called 'environmental,' in that, e.g., figure A is separated into subfigures on the information that these subfigures are present, already separated, in figure B. An interesting extension to the present part I of ANALOGY might be to incorporate some form of 'intrinsic'

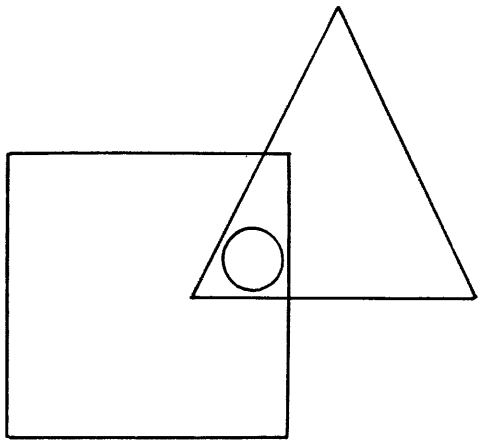


Figure 4a.

decomposition in which 'most plausible' decompositions are generated according to Gestalt-like criteria of 'good figure.' Such an extension could widen the problem-solving scope of ANALOGY considerably to include many cases where the appropriate subfigures do not appear already 'decomposed' among the problem figures. For example, suppose problem figures A and B are as shown in Figs. 4a and 4b, respectively. A decomposition into the square, triangle, and circle seems necessary to state a reasonable transformation rule. This example, incidentally, illustrates one potentially useful 'intrinsic' decomposition heuristic: roughly, choose decompositions into subfigures which have as much internal symmetry (in some precise sense) as possible.

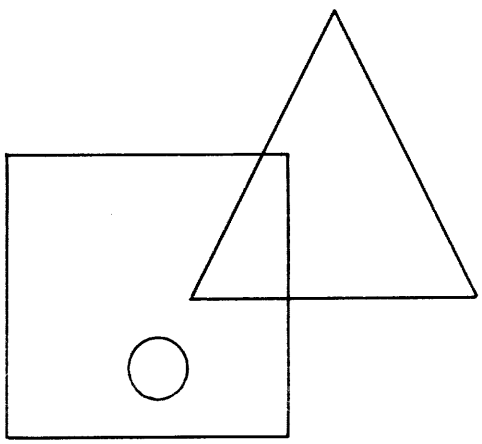


Figure 4b.

Next, the 'objects' generated from the decomposition process are given to a routine which calculates a specified set of properties of these objects and relations among them. The program is designed so that this set can be changed easily. As a sample of a relation-calculating subroutine, we cite one that calculates, for figure A of our example, that the object labeled P2 lies inside that labeled P3 and generates a corresponding expression (INSIDE P2 P3) to be added to the part I output description of figure A. The method used involves calculating all intersections with P3 of a line segment drawn from a point on P2 to the edge of the field (all figures are considered as drawn on a unit square). In this case P2 lies inside P3 since the number of such intersections is odd, namely one (and P3 is known to be a simple closed curve—if it were not, the calculation just described would be performed for each closed curve contained in P3). To do this, a substantial repertoire of 'analytic geometry' routines is required for part I, to determine, for example, intersections of straight line segments and arcs of circles in all cases and combinations. Other relation routines available in part I calculate, for example, that in figure A of our example P1 is above P2 and P3 and in figure B that P4 is to the left of P5.

The principal business of part I, aside from decomposition and the property and relation calculations, is a set of 'similarity' calculations. Here, part I determines, for each appropriate pair of objects, all members from a certain class T of transformations which carry one object of the pair into the other. The elements of T are compositions of Euclidean similarity transformations (rotation and uniform scale change) with horizontal and vertical reflections. Given descriptions of virtually any pair of arbitrary line-drawings x and y, the routines of part I will calculate the parameters of all instances of transformations from T that 'map' x into y.

More precisely, an acceptable 'map' is a member of T for which T(x) is congruent to y up to certain metric tolerances which are parameters in the corresponding programs.

This routine is, in effect, a pattern-recognition program with built-in invariance under scale changes, rotations, and certain types of reflections. It consists essentially of a topological matching process, with metric comparisons being made between pairs of lines selected by the topological process. In Ref. 6 Sherman introduced some topological classification into a sequential decision tree program for the recognition of hand-printed letters, but the notion of systematically using the topological information to determine which metric comparisons are to be made seems to be new. This type of organization for pattern recognition has its own advantages (e.g., flexibility—the metric parts can be changed easily with no effect on the overall structure) and difficulties (e.g., sensitivity to metrically small changes in a figure which affect the connectivity—but this sensitivity can be largely removed by suitable pre-processing). Incidentally, it may be worth noting that if we suppress the metric comparisons entirely we get a general, and reasonably efficient, topological equivalence algorithm for graphs (networks).

The set of techniques we have just been describing, based on the use of a list-processing language to perform processing of line drawings by manipulating their list-structured descriptions, is by no means limited in applicability to the uses to which we have put it in part I of ANALOGY. To the contrary, it is our view that the representation of line drawings used here and the corresponding processing routines form a suitable basis for the development of a quite powerful 'line-drawing-manipulation language' with potential usefulness in a wide variety of applications. Regardless of whether the present investigation turns out to have a measurable influence on the art of designing problem-solving programs, it seems probable that the principal short-range contribution of ANALOGY is in the picture-processing by-products just described. (Incidentally, these techniques were discussed briefly from an ANALOGY-independent point of view in Ref. 17.)

After the similarity information is computed for every required pair of objects, both within a problem figure and between figures, this information, together with the decomposition and property and relation information, is punched out on cards in a standard format for input to part II. (For a typical set of figures, the total output of part I, punched at up to 72 columns/card, might come to 15 to 20 cards.)

Part II is given these cards as input. Its final output is either the number of the solution figure or a statement that it failed to find an answer. Its first step is to generate a rule (or, more frequently, a number of alternate rules) transforming figure A into figure B. Such a rule specifies how the objects of figure A are removed, added to, or altered in their properties and their relations to other objects to generate figure B. Once this set of rule possibilities has been generated, the next task is to 'generalize' each rule just enough so that the resulting rules still take figure A into figure B and now take figure C into exactly one of the answer figures. More precisely, for each 'figure A → figure B' rule and for each answer figure, part II attempts to construct a 'common generalization' rule which both takes figure A into figure B and figure C into the answer figure in question. This process may produce a number of rules, some very weak in that virtually all the distinguishing detail has been 'washed out' by 'generalization.' Hence it is necessary at this point to pick the 'strongest' rule by some means. This entire process requires a complex mechanism for manipulating and testing the rules and deciding which of the several rule candidates, the results of different initial rules or of different 'generalizations,' is to be chosen.

The principal method embodied in part II at present is able to deal quite generally with problems in which the numbers of parts added, removed, and matched in taking figure A into figure B are the same as the numbers of parts added, removed, and matched, respectively, in taking figure C into the answer figure. A substantial majority of the questions on the tests we have used are of this type, as is our present example; virtually all would be under a sufficiently elaborate decomposition process in part I; this restriction still permits a wide variety of transformation rules. It should be mentioned

that the methods of part II have been kept subject-matter free; no use is made of any geometric meaning of the properties and relations appearing in the input to part II.

The more detailed workings of both parts I and II are best introduced through examining the process sketched above at work on our example. To convey some further feeling for the nature of the input to part I, we exhibit part of it, namely, the input description of figure A. The LISP expressions look like:

```
(
(DOT (0.4 . 0.8))
(SCC ((0.3 . 0.2) 0.0 (0.7 . 0.2) 0.0 (0.5 .
    0.7) 0.0 (0.3 . 0.2)))
(SCC ((0.4 . 0.3) 0.0 (0.6 . 0.3) 0.0 (0.6 .
    0.4) 0.0 (0.4 . 0.4) 0.0 (0.4 . 0.3)))
)
```

The first line above corresponds to the dot (at coordinates x = 0.4 and y = 0.8 on the unit square; the coordinate pairs in the other expressions are interpreted analogously). The next two lines correspond to the triangle (SCC stands for simple closed curve. All connected figures are divided into three classes—dots (DOT), simple closed curves (SCC), and all the rest (REG). This is solely for reasons of programming convenience; no other use is made of this three-way classification). Each non-connected figure is represented simply by a list of descriptions of its connected parts.

A curve (which may consist of an arbitrary sequence of elements chosen from straight line segments and arcs of circles) is represented by a list in which coordinate pairs alternate with the curvatures of the line elements between (all curvatures are zero here since the lines in question are all straight). Similarly, the next two lines above correspond to the rectangle; the entire description of figure A is then a list of the descriptions of these three parts. The format corresponding to the non-SCC figures like the Z-shaped subfigure of figure C is similar though somewhat more complex; it looks like:

```
(REG ((V1 V2 (0.0 (0.55 . 0.5) 0.0 (0.45 .
    0.3) 0.0))
  (V2 V1 (0.0 (0.45 . 0.3) 0.0 (0.55 . 0.5)
    0.0))))
```

where V1 and V2 are the two vertices (here, endpoints) of the figure. The coordinates of V1 and V2 are given to part I in a separate list. They are V1 = (0.45 . 0.5), V2 = (0.55 . 0.3). Here, the top-level list describes the connectivity by stating which vertices are connected to which and how often—sublists describe in detail the curves making these connections. (By vertex we mean either an endpoint of a curve or a point at which three or more curves come together.) The complete details of the input format are given in Ref. 1, along with many examples.

When the input shown above corresponding to problem figure A and the corresponding inputs for the other seven figures are processed, the output from part I is, in its entirety, the ten LISP expressions shown below. For brevity, all similarity information concerning non-null reflections has been deleted. Also, we have replaced the actual arbitrary symbols generated internally by ANALOGY as names for the parts found by the decomposition program by the names P1, P2, etc., which appear as labels on our example figures above. The ten output expressions are:

(1) ((P1 P2 P3) . ((INSIDE P2 P3) (ABOVE P1 P3) (ABOVE P1 P2)))

(2) ((P4 P5) . ((LEFT P4 P5)))

(3) ((P6 P7 P8) . ((INSIDE P7 P6) (ABOVE P8 P6) (ABOVE P8 P7)))

(4) ((P2 P4 (((1.0 . 0.0) . (N.N)) ((1.0 . 3.14) . (N.N)))) (P3 P5 (((1.0 . 0.0) . (N.N)))))

(5) ((P1 P8 (((1.0 . 0.0) . (N.N)))))

(6) NIL

(7) ((P9 P10 P11) (P12 P13) (P14 P15) (P16 P17) (P18))

(8) (((INSIDE P10 P9) ABOVE P11 P9) (ABOVE P11 P10)) ((LEFT P12 P13)) ((INSIDE P15 P14)) ((ABOVE P17 P16)) NIL)

(9) (((P6 P9 (((1.0 . 0.0) . (N.N)))) (P7 P10 (((1.0 . 0.0) . (N.N)) ((1.0 . —3.14) . (N.N)))) (P8 P11 (((1.0 . 0.0) . (N.N)))))

```
((P6 P13 (((1.0 . 0.0) . (N.N))))
  (P7 P12 (((1.0 . 0.0) . (N.N))
  ((1.0 . —3.14) . (N.N)))))
((P6 P14 (((1.0 . 0.0) . (N.N))))
  (P7 P15 (((1.0 . 0.0) . (N.N))
  ((1.0 . —3.14) . (N.N)))))
((P6 P16 (((1.0 . 0.0) . (N.N))))
  (P8 P17 (((1.0 . 0.0) . (N.N)))))
((P7 P18 (((1.0 . 0.0) . (N.N)) ((1.0 .
  —3.14) . (N.N.))))) )
(10) ( ( ((P1 P11 (((1.0 . 0.0) . (N.N)))))
  NIL NIL
  ((P1 P17 (((1.0 . 0.0) . (N.N)))))
  NIL ) . (NIL NIL NIL NIL NIL) )
```

To explain some of this: The first expression corresponds to figure A. It says figure A has been decomposed into three parts, which have been given the names P1, P2, and P3. Then we have a list of properties and relations and similarity information internal to figure A, namely, here, that P2 is inside P3, P1 is above P2, and P1 is above P3. The next two expressions give the corresponding information for figures B and C. The fourth expression gives information about Euclidean similarities between figure A and figure B. For example, P3 goes into P5 under a 'scale factor = 1, rotation angle = 0, and both reflections null' transformation. The next two expressions contain the corresponding information between figure A and figure C and between figure B and figure C, respectively. The seventh list is a five-element list of lists of the parts of the five answer figures; the eighth a five-element list of lists, one for each answer figure, giving their property, relation, and similarity information. The ninth is again a five-element list, each a 'similarity' list from figure C to one of the answer figures. The tenth, and last, expression is a dotted pair of expressions, the first again a five-element list, a 'similarity' list from figure A to each of the answer figures, the second the same from figure B to each of the answer figures. This brief description leaves certain loose ends, but it should provide a reasonably adequate notion of what is done by part I in processing our sample problem.

The ten expressions above are given as arguments to the top-level function of part II (optimistically called *solve*). The basic method employed by *solve*, which suffices to do this problem, begins by matching the parts of figure A and those of figure B in all possible ways compatible with the similarity information. From this process, it concludes, in the case in question, that P2 → P4, P3 → P5, and P1 is removed in going from A to B. (The machinery provided can also handle far more complicated cases, in which alternate matchings are possible and parts are both added and removed.) On the basis of this matching, a statement of a rule taking figure A into figure B is generated. It looks like:

```
(
(REMOVE A1 ((ABOVE A1 A3) (ABOVE
    A1 A2) (SIM OB3 A1 (((1.0 . 0.0) .
    (N.N))))))
(MATCH A2 (((INSIDE A2 A3) (ABOVE
    A1 A2) (SIM OB2 A2 (((1.0 . 0.0) .
    (N.N))))) . ((LEFT A2 A3) (SIM
    OB2 A2 (((1.0 . 0.0) . (N.N)) ((1.0 .
    3.14) . (N.N)))) (SIMTRAN (((1.0 .
    0.0) . (N.N)) ((1.0 . 3.14) . (N.N)
    ))))))
(MATCH A3 (((INSIDE A2 A3) (ABOVE
    A1 A3) (SIM OB1 A3 (((1.0 . 0.0) .
    (N.N))))) . ((LEFT A2 A3) (SIM
    OB1 A3 (((1.0 . 0.0) . (N.N))))
    (SIMTRAN (((1.0 . 0.0) . (N.N)
    ))))))
)
```

The A's are used as 'variables' representing objects. The format is rather simple. For each object added, removed, or matched, there is a list of the properties, relations and similarity information pertaining to it. (In the case of a matched object, there are two such lists, one pertaining to its occurrence in figure A and the other to its occurrence in figure B.) There are two special devices; the (SIM OB1 . . .) — form expressions give a means of comparing types of objects between, say, figure A and figure C; the other device is the use of the SIMTRAN expressions in the figure-B list for each matched object. This enables us to handle conveniently some additional situations which we shall omit from consideration, for brevity. They are treated in detail in Ref. 1.

The above rule expresses everything about figures A and B and their relationship that is used in the rest of the process. (The reader may verify that the rule does, in some sense, describe the transformation of figure A into figure B of our example.)

Next, a similarity matching is carried out between figure C and each of the five answer figures. Matchings which do not correspond to the ones between figure A and figure B in numbers of parts added, removed, and matched, are discarded. If all are rejected this method has failed and *solve* goes on to try a further method. In the present case, figures 1 and 5 are rejected on this basis. However, figures 2, 3, and 4 pass this test and are examined further, as follows. Choose an answer figure. For a given matching of figure C to the answer figure in question (and *solve* goes through all possible matchings compatible with similarity) we take each 'figure A → figure B' rule and attempt to fit it to the new case, making all matchings between the A's of the rule statement and the objects of figure C and the answer figures which are compatible with preserving add, remove, and match categories, then testing to see which information is preserved, thus getting a new, 'generalized' rule which fits both 'figure A → figure B' and 'figure C → the answer figure in question.' In our case, for each of the three possible answer figures we get two reduced rules in this way (since there are two possible pairings between A and C, namely, P1 ⟷ P8, P2⟷P6, and P3 ⟷ P7, or P1 ⟷ P8, P2 ⟷ P7, and P3 ⟷ P6).

In some sense, each of these rules provides an answer. However, as pointed out earlier, we want a 'best' or 'strongest' rule, that is, the one that says the most or is the least alteration in the original 'figure A → figure B' rule and that still maps C onto exactly one answer figure. A simple device seems to approximate human opinion on this question rather well; we define a rather crude 'strength' function on the rules and sort them by this. If a rule is a clear winner in this test, the corresponding answer figure is chosen; if the test results in a tie, the entire method has failed and *solve* goes on to try something else. In our case, when the values for the six rules are computed, the winner is one of the rules corresponding to figure 2, so the program, like all humans so far consulted, chooses it as the answer. The rule chosen looks like this:

```
(
(REMOVE A1 ((ABOVE A1 A3) (ABOVE
    A1 A2) (SIM OB3 A1 (((1.0 . 0.0) .
    (N.N))))))
(MATCH A2 (((INSIDE A2 A3) (ABOVE
    A1 A2)) . ((LEFT A2 A3) (SIMTRAN
    (((1.0 . 0.0) . (N.N)) ((1.0 . 3.14) .
    (N.N)))))))
(MATCH A3 (((INSIDE A2 A3) (ABOVE
    A1 A3)) . ((LEFT A2 A3) (SIMTRAN
    (((1.0 . 0.0) . (N.N)))))))
)
```

Again, it is easy to check that this rule both takes figure A into figure B and figure C into figure 2, but not into any of the other answer figures.

Further Examples and Comments

(a) *Examples*

We first exhibit several additional examples of problems given to ANALOGY:

(i) (See Fig. 5)

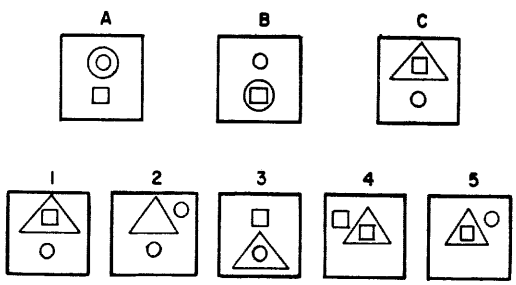


Figure 5.

Here the rule involves changes in the relations of the three parts. ANALOGY chose answer figure 3.

(ii) (See Fig. 6)

This case involves both addition and removal of objects. ANALOGY chose answer figure 2.

(iii) (See Fig. 7)

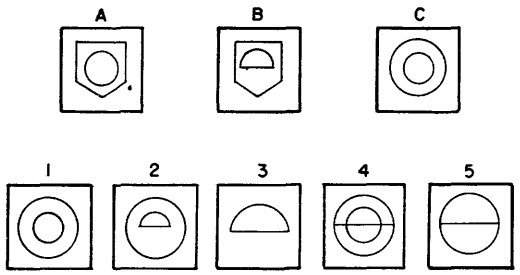


Figure 6.

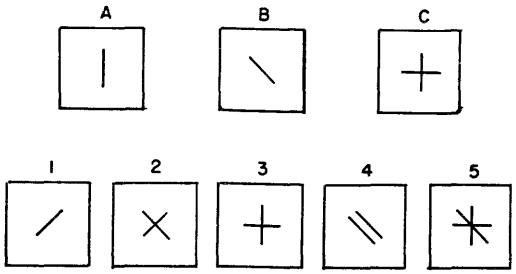


Figure 8.

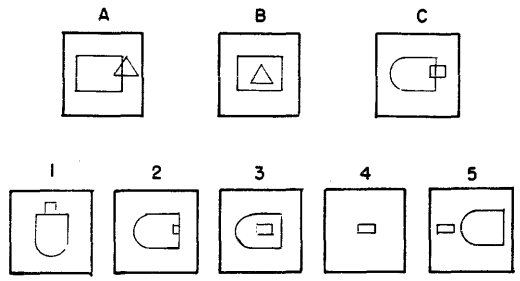


Figure 7.

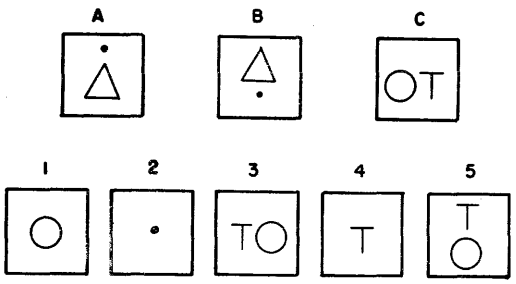


Figure 9.

Note that this case required the more powerful decomposition program. Here ANALOGY chose figure 3.

(iv) (See Fig. 8)

The rule here simply involved a rotation. ANALOGY chose figure 2.

(v) (See Fig. 9)

Here ANALOGY chose figure 3, using an extension of the part II techniques discussed above. This extension, employed after failure of the basic process, involves systematic substitution of certain specified relations (e.g., LEFT for ABOVE) for others in the part II input descriptions, thus making it possible for ANALOGY to relate the 'vertical' transformation taking A into B to the 'horizontal' transformation of C into 3.

(vi) In the problem of Fig. 1, the large circle of answer figure 4 was replaced by a large square and the problem rerun. Again figure 4 was chosen but by a different rule. Now, instead of the inner object being removed, as before, the outer object is removed and the inner one enlarged. This illustrates some of the flexibility of the procedure and the dependence of the answer choice on the range of allowed answers as well as on A, B, and C.

(vii) (See Fig. 10)

Here is an example of a failure by ANALOGY to agree with the human consensus which favors figure 5. ANALOGY chose figure 3.

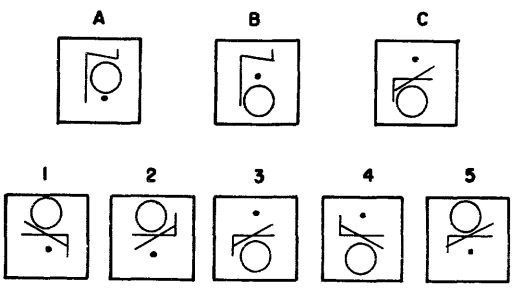


Figure 10.

(b) *Comparison with Human Performance*

We can only roughly compare the performance of ANALOGY with that of humans on geometric-analogy problems, since ANALOGY has not yet been given the complete set of such problems from any test for which scores are available. However, as some indication, we cite scores on the ACE tests based on a period of years including those editions of the test from which most of the problems on which ANALOGY was tested were selected. These scores are for a large population of college-preparatory students; the median score, on a test consisting of 30 such questions, ranged from 17 for 9th grade to 20 for 12th grade. Our estimate is that, on the same tests, ANALOGY, as it currently exists, could solve between 15 and 20 problems. Given, in addition, certain changes (mostly in part I, e.g., a more powerful decomposition program and additional properties and relations) for which we have reasonably well-worked-out implementations in mind, ANALOGY should be capable of perhaps 25 successful solutions.

(c) *The Use of LISP*

The use of a list-processing language to construct the ANALOGY program appears to have been a suitable choice; most notably, its capability at handling intermediate expressions of unpredictable size and 'shape' (such as our figure descriptions and transformation rules) is of great value. We especially wish to praise LISP as a convenient and elegant language in which to write and debug complex programs. The ease of composition of routines, the highly mnemonic nature of the language, and the good tracing facilities all contribute greatly to effective program construction. In return for the use of such a language one pays a certain price in speed and storage space, which, in the case of ANALOGY, at least, was a very acceptable bargain, since the necessity of machine-language coding would have made the entire project unfeasible. Incidentally, the ANALOGY program (apparently the largest program written in LISP to date) is so large that parts I and II must occupy core separately. The consequent limited (and one-way) communication between the parts was a serious design constraint but proved to have some compensating advantages in simplicity.

ANALOGY and Pattern-Recognition in Problem-Solving Programs

In this section we shall consider certain aspects of the design of problem-solving machines. To aid this discussion we shall specify (rather loosely) a subclass of problem-solving machines and carry out our discussion in terms of these though the ideas involved are by no means limited in applicability to this class. The machines we have in mind are typified by GPS[14] in that the problem to be solved by the machine is to transform one specified 'object' or 'situation' (whatever this may mean in a particular subject-matter context) into another by applying an appropriate sequence of transformations chosen from a class available to the machine. A wide variety of problems may be cast in this form (again see Ref. 14 or other discussions of GPS by the same authors). As in GPS, subgoals may be generated and attacked by such a machine and elaborate schemes of resource allocation may be required. However, these aspects do not concern us here. Our interest lies in the basic task of the machine; given a pair of 'objects,' it must choose an 'appropriate' transformation, i.e., one contributing to the goal of transforming one of the given 'objects' into the other.

It is a widely-held view, with which we agree completely, that for a machine to be capable of highly intelligent behavior on a task of this kind, in a rich environment of objects and transformations (and, in particular, to be capable of learning at a level more advanced than that of present machines), the critical factor is that it have a good internal representation of both its subject matter ('objects') and its methods ('transformations'), as well as an elaborate set of 'pattern-recognition' techniques for matching transformations to object pairs. Probably this means a quite 'verbal' representation of both objects and transformations as expressions in suitable 'description languages.' Furthermore, these matching techniques must be represented in a form in which they themselves are capable of being improved as the machine gains experience. The central role which 'pattern-recognition' techniques must play in sophisticated problem-solving programs and the corresponding importance for effective learning of autonomous improvement in the perform-

ance of these techniques are well expressed in Minsky.[12] There we find:

In order not to try all possibilities a resourceful program must classify problem situations into categories associated with the domains of effectiveness of the machine's different methods. These pattern-recognition methods must extract the heuristically significant features of the objects in question. Again from Ref. 12 we have:

Again from [12] we have:

In order to solve a new problem one uses what might be called the basic learning heuristic—first try using methods similar to those which have worked, in the past, on similar problems.

Here, the problem is, of course, to have pattern-recognition techniques possessing, or able themselves to learn, criteria of 'similarity' appropriate to the subject matter in question.

The 'fixed-length property-list' schemes (see Ref. 12) which characteristically have been used to perform this pattern-recognition task in current problem-solving programs have two principal defects which limit their extension to harder problems:

(i) While, in principle, given enough sufficiently elaborate properties, one can make arbitrarily fine discriminations, in practice a given set of properties will begin to fail rapidly as situations become more complex. In particular, for 'situations' which must be treated as consisting of interrelated parts, the 'global' nature of the scheme in question leaves it helpless.

(ii) Such a scheme is very limited in its learning capabilities, since it has access to very little information about its component properties; in particular, it is incapable of "knowledgeably' modifying its tests or adding new ones—it can only modify the weightings given to the results of these tests in its 'decisions.'

In view of the limitations of the 'property-list' pattern-recognition scheme just mentioned, we can formulate some requirements for a pattern-recognition scheme suitable tc replace it as a 'transformation-selecting' mechanism. First, the scheme must have access to a representation of each 'object' in terms of a 'descriptive framework' for the subject matter in question which is suitable in that useful relationships between 'objects' can be extracted relatively simply from the corresponding representations. Furthermore, the transformation-selecting rules of the pattern-recognition apparatus should themselves be expressed in a representation suitable for a 'learning mechanism' to revise the set of rules (i) by adding new rules and deleting those old ones which prove least useful as experience associates certain object pairs with certain transformations and (ii) by replacing a set of particular rules by a 'common generalization' rule again represented in the same language. Such facilities could go far toward removing the limitations of which we have spoken and providing both a powerful rule language (the rules can be stated in terms of the 'descriptive framework' we have postulated for the 'objects') and a learning mode more sophisticated than any yet incorporated in such a general problem-solving program.

So far we have been enumerating desirable features in a 'pattern-recognition' mechanism to be used as a transformation-selection device within a large problem-solver. What has all this to do with ANALOGY, which is not even a problem-solving program of the class we have been considering? We suggest that ANALOGY can, under a suitable (rather drastic) reinterpretation, be to some extent viewed as a pattern-recognition program having, to the limited degree appropriate for its particular environment, all the features we have listed. First, the 'objects' are the problem figures of ANALOGY and the suitable 'descriptive framework' appropriate to these objects is the 'subfigure and relation' representation used as the input part I generates for part II of ANALOGY. (Thus part I of ANALOGY corresponds to the apparatus that generates this representation for each object; that is, it goes from a representation of the 'problem objects' which is convenient for input to the problem-solver to one which is in a form suitable for internal use.) The generation in ANALOGY of a transformation rule taking one answer figure into another can be thought of as corresponding to the first kind of learning we listed above, namely, the adding of rules as, with experience, the machine associates certain object pairs with certain simple or com-

posite transformations. Finally, the common generalization of two rules in ANALOGY corresponds to the second kind of learning we mentioned, namely, the generation of a common generalization of several rules associating 'objects' and 'transformations.' Furthermore, ANALOGY's process of choosing between 'common generalizations' of different rule pairs mirrors a process of selectively incorporating only those generalizations with the greatest discriminatory power. Under this interpretation, ANALOGY appears as a model for a pattern-recognition process with all the characteristics mentioned. The potential value of ANALOGY, viewed in this way, as a suggestive model for the construction of such pattern-recognition mechanisms for use within problem-solving programs may prove to be the chief product of our work with ANALOGY and the best justification for having carried it out.

References

1. T. G. EVANS, PH.D. Thesis, Department of Mathematics, MIT, June, 1963 (soon to be available as an AFCRL Technical Report).
2. J. MCCARTHY, "Recursive functions of symbolic expressions," Comm. ACM, Vol. 3, April, 1960.
3. J. MCCARTHY *et al.*, LISP 1.5 Programmer's Manual, MIT, revised edition, August, 1962.
4. R. L. GRIMSDALE, F. H. SUMNER, C. J. TUNIS, and T. KILBURN *et al.*, "A system for the automatic recognition of patterns," Proc. IEE, March, 1959, Vol. 106, pt. B, pp. 210-221.
5. T. MARILL, A. K. HARTLEY, T. G. EVANS, B. H. BLOOM, D. M. R. PARK, T. P. HART, and D. L. DARLEY, "CYCLOPS-1: a second-generation recognition system," FJCC, Las Vegas, Nevada, November, 1963.
6. H. SHERMAN, "A quasi-topological method for the recognition of line patterns," Proc. ICIP, Paris, France, June, 1959, pp. 232-238.
7. I. SUTHERLAND, "Sketchpad: a man-machine graphical communication system," SJCC, Detroit, Michigan, May, 1963.
8. L. ROBERTS, PH.D. Thesis, Department of Electrical Engineering, MIT, June, 1963.
9. R. KIRSCH, personal communication.
10. L. HODES, "Machine processing of line drawings," Lincoln Laboratory Technical Memorandum, March, 1961.
11. R. CANADAY, M.S. Thesis, Department of Electrical Engineering, MIT, February, 1962.
12. M. L. MINSKY, "Steps toward artificial intelligence," Proc. IRE, January, 1961, pp. 8-30.
13. A. NEWELL and H. A. SIMON, "The logic theory machine," IRE Trans. on Information Theory, Vol. IT-2, #3, September, 1956, pp. 61-79.
14. A. NEWELL, J. C. SHAW, and H. A. SIMON, "Report on a general problem-solving program," Proc. ICIP, Paris, France, June, 1959, pp. 256-264.
15. H. GELERNTER and N. ROCHESTER, "Intelligent behavior in problem-solving machines," IBM J. Res. and Dev., Vol. 2, #4, October, 1958, pp. 336-345.
16. J. SLAGLE, PH.D. Thesis, Department of Mathematics, MIT, June, 1961.
17. T. G. EVANS, "The use of list-structured descriptions for programming manipulations on line drawings," ACM National Conference, Denver, Colorado, August, 1963.

Acknowledgements

The assistance of the Cooperative Test Division of the Educational Testing Service, Princeton, New Jersey, in providing a large set of geometric-analogy questions from its files is gratefully acknowledged.

Thanks are also due to the Educational Records Bureau, New York, N.Y., for the statistics on human performance on geometrical-analogy questions cited in Sec. 4b.

Most of the computation associated with the development and testing of ANALOGY was performed at the MIT Computation Center.

EXPERIMENTS WITH A THEOREM-UTILIZING PROGRAM

Larry E. Travis
System Development Corporation
Santa Monica, California

1. SIGNIFICANCE OF THE THEOREM-UTILIZING PROBLEM

(1.1) *Computers as Expert Problem Solvers*

There are a large number of difficult intellectual tasks which consist of synthesizing a partial ordering. To program a computer we must synthesize a sequence of computer instructions; one way of constructing a logical derivation is by synthesizing a sequence of inference-rule applications which transform given premises into desired conclusion; one way of choosing among alternative moves in a game is by synthesizing and evaluating trees of possibly ensuing moves. Can we program a computer to perform such tasks? Can we program a computer to perform such tasks expertly, i.e., well enough that it can by itself surpass a skilled human being in the performance of such tasks?

A decade of work on the subject by Gelernter, Minsky, Newell, Samuel, Shaw, Simon, Slagle, Tonge, and others has given us a clear *yes* in answer to the first question. But the answer to the second is still very much in doubt. The comparisons so far have been with Navy boots, high school sophomores, and college freshmen. The question at issue concerns the difference between performing a sequence-synthesis task well and performing it poorly if at all. We would ask: What is skill in tasks like theorem proving and program writing? Suppose we have two machines each of which is able to prove at least some theorems but one rather like an unskilled student and other rather like a skilled mathematician. How do they differ?

(1.2) *Synthesizing Sequences of Symbol Transformations*

We must be a little more specific about the kind of task we are interested in. In logic, constructing a derivation is constructing a sequence of permissible symbol transformations by which given premises can be transformed into desired conclusions. The permissible symbol transformations are the basic rules of inference. Thus in a system where the permissible transformations include detachment (i.e., transforming P and $P \rightarrow Q$ into Q), interchange of conditionals and disjunctions (i.e., transforming $P \rightarrow Q$ into $\sim P \; v \; Q$ and $P \; v \; Q$ into $\sim P \rightarrow Q$), and commutation of disjunctions (i.e., transforming $P \; v \; Q$ into $Q \; v \; P$), an example of a derivation is the following:

GIVEN PREMISES:

(1) $P \rightarrow (Q \rightarrow R)$
(2) P
(3) $\sim R$

DESIRED CONCLUSION: $\sim Q$

DERIVATION STEPS:

(4) $Q \rightarrow R$	Detachment transformation of Premises (1) and (2)
(5) $\sim Q \; v \; R$	Interchange transformation of Step (4)
(6) $R \; v \; \sim Q$	Commutation transformation of Step (5)

DERIVATION STEPS—Continued

(7)	$\sim R \rightarrow \sim Q$	Interchange transformation of Step (6)
(8)	$\sim Q$	Detachment transformation of Premise (3) and Step (7)

Similarly in computing, constructing a program is constructing a sequence of available symbol transformations by which given input can be transformed into desired output. The available symbol transformations are the primitive computer instructions. Similar characterizations can be offered for game playing, puzzle solving, manipulating algebraic expressions, and other tasks. There are, of course, important differences among these tasks, especially in the way the tasks are defined for the problem solver, but for the present our concern is with what they have in common.

(1.3) *Skill at Sequence Synthesis*

It would appear that one thing most sequence-synthesis tasks have in common, at least insofar as human problem solvers are concerned, is a way in which skill at their performance is acquired. Becoming a skilled logician or a skilled programmer mainly consists in learning useful combinations of the available primitive transformations and in developing a sensitivity to the occasions on which it is appropriate to apply these combinations. As we shall see, there is more to it than this, but this is a good place to start.

Some examples will clarify our conception of skill. The easiest examples to come by are those where the skilled human being takes explicit note of a useful combination of primitives. Thus the logician adds defined rules of inference to his system. Or he just simply adds a theorem to his list of established statements. For instance, very early in the development of a system for the propositional calculus the logician will notice the usefulness of the transposition theorem $(Q \rightarrow R) \rightarrow (\sim R \rightarrow \sim Q)$ and, after proving it with a sequence something like Steps (4) through (7) of our example above, will add it to his list of axioms and other proved theorems. Henceforth in developing a derivation, whenever he needs to transform an expression whose structure is like the antecedent of the theorem into an expression whose structure is like the consequent of the theorem, he merely cites the theorem. He does not indicate, as is done in the example above, all the primitive transformations, the primitive rules of inference, which are required actually to effect the transformation. Theorems are usually thought of as additions to the stock of assertions rather than as additions to the stock of transformation rules of a logical system. But we shall be primarily interested in them in their role as explicit representations of defined, complex transformations, i.e., transformations which can be effected by combinations of primitives.

In programming, the defined, complex transformations are the subroutines and macroes, some explicitly formulated, labelled, and stored on tape but many more, accumulated in the experience of writing many programs of many different kinds, "stored" more or less completely only in the programmer's head.

(1.4) *Construction Problems and Existence Problems*

An important distinction is to be noticed between the kinds of sequence-synthesis problems represented by programming problems on the one hand and by theorem-proving problems on the other. The programmer has to produce the actual sequence of primitive transformations while the logician does not. This arises from a difference in the reasons why the two are interested in synthesizing sequences of primitive transformations. In logic the question of interest is whether such a sequence connecting premises and conclusion exists at all; if it can be shown to exist the logical problem is settled, and though the logician may be worried about problems of elegance and efficiency, these are extralogical worries. The programmer, on the other hand, must synthesize an actual sequence of primitives. There is usually no question of whether such a sequence exists; the programming requirement is production of one to do some work. The programmer needs the actual sequence, not just the knowledge that one exists.

We shall refer to the programmer's problems as construction problems, opposing them to the logician's existence problems. The distinction is

not a neat one because the logician usually uses methods such that from a demonstration of existence of a sequence of primitive inference rules he could automatically (and tediously) produce an actual sequence if called upon to do so. But the distinction is worth noting because it is construction problems with which we shall be primarily concerned in the present paper.

(1.5) *Related Research*

There has been a large amount of work in the recent past concerned with programming computers to prove theorems (e.g., Wang,[42] Gilmore,[7] Davis and Putnam,[3] and Robinson[19]) and with programming computers to program themselves (e.g., Kilburn *et al.*[9] and Amarel).[1] Though several of these authors, e.g., Wang and Amarel, respectively, have commented on the apparent importance of a mechanical theorem prover's being able to save and utilize previously-proved theorems and of a self-programming machine's being able to save and utilize previously-developed subroutines, there has been little work directly on the problems raised. Programs with at least some ability to utilize previously-proved theorems (or their counterparts) are Newell, Shaw, and Simon's Logic Theorist,[15,16] Gelernter's Geometry Theorem Prover,[5,6] Slagle's Automatic Integrator,[22] and Simon's Heuristic Compiler.[21] We would suggest that it is because it attempts to prove theorems by using previously-proved theorems that the Logic Theorist is of special importance even though it turns out to be a rather weak theorem prover in comparison with other systems which use alternative formulations and methods. Wang avers that Newell, Shaw, and Simon's approach to proving theorems by machine is rather like trying to kill a chicken with a butcher knife. It is true that there are more efficient ways to dispatch a chicken, but there may be no better way to learn how to use a butcher knife.

We turn, then, to reporting an investigation concerned not with theorem-proving or self-programming machines *per se* but with the more specific problem of how such sequence-synthesizing machines might be made efficiently to exploit previously-developed sequences when later given a problem of constructing a more complex sequence. Our approach has been actually to write and run what can reasonably be called a theorem-utilizing program. We shall first describe the program. Then we shall report some results obtained with it and some conclusions to be drawn from these results. Finally, we shall indicate what appears to be a reasonable direction for future research on the problem.

2. DESCRIPTION OF A THEOREM-UTILIZING PROGRAM

(2.1) *Criteria of Program Evaluation*

Our goal has been to construct an automatic theorem-proving system which proves theorems by efficiently utilizing previously-proved theorems. The criterion of success is realization of a theorem prover which becomes progressively better as it accumulates and otherwise modifies its store of previously-proved theorems. It should be remarked that improvement of the store is not merely a matter of accumulation and, as a matter of fact (as clearly demonstrated by Newell, Shaw, and Simon's[18] work), under certain conditions simple increase in size of the store can hinder rather than help problem-solving ability. This ability is more a matter of what particular previously-proved theorems are available and what information is stored about them and how this information is used, than it is a matter of how many are available. It is specifically to these points of how to select useful theorems for remembering, how to abstract them and their proofs, and how to use the abstracted information that the research here reported has been directed.

(2.2) *The Program's Problem Domain*

For our experiments with theorem proving we devised a special problem domain rather than selecting some standard domain such as the propositional calculus, group theory, or plane geometry. We shall presently give our reasons for doing this, but first let us describe the domain. It will probably be simpler if we first present problems in the domain as if they were problems of programming a very simple computer best characterized as a vector adder. The computer has no data memory. It is given as input an ordered n-tuple of numbers and operates on this n-tuple according to a given

program. The program is a simple sequence with each of its constituent basic instructions operating directly on the output of the immediately preceding instruction in the sequence. Each of the basic instructions adds a quantity (possibly zero or negative) to each term of the n-tuple on which it operates. Thus each of these instructions can also be represented as an ordered n-tuple of numbers.

Consider an example of a 4-tuple adder which has as its basic instructions (S)<0,1,2,3>, (T)<3,2,1,0>, and (W)<0,—1,1,0>. A typical programming problem might be that of synthesizing a sequence of these instructions which will transform <4,9,2,7> into <10,10,10,10>. As can easily be verified, one program which will do this is *(S), (T), (T), (W), (W), (W), (W)*. The first instruction (S) transforms <4,9,2,7> into <4,10,4,10>, the second instruction (T) transforms this into <7,12,5,10>, etc., until operation of the final instruction (W) gives an output of <10,10,10,10>.

Problems of the kind we are considering can just as well be interpreted as theorem-proving problems in what might be called a tally calculus. Thus, rather than the n-tuple <4,9,2,7> we might have the expression

AAAA/BBBBBBBBB/CC/DDDDDDD

and we might ask whether

AAAAAAAAAA/BBBBBBBBBB/
CCCCCCCCCC/DDDDDDDDDD

can be derived from it given basic rules of inference corresponding to (S), (T), and (W). These rules are to be interpreted in such a way that (W), for instance, means: make no change in the *A*'s, delete one *B*, add one *C*, and make no change in the *D*'s.

In such a tally calculus there might well be existence problems as well as construction problems. Most of the calculi with which we have actually worked, however, have been supplied with basic rules of inference sufficient to make possible the derivation of any expression in the calculus from any other expression. Our reason for choosing such calculi has been that we are interested in problem-solving mechanisms and not in whether some particular derivations are possible, and we learn little from applying these mechanisms to problems which have no solution. It will be a very long time before we can hope to have non-algorithmic problem-solving mechanisms powerful enough that their failure to discover a solution is presumptive evidence that no solution exists.

We have imposed on our problem solver requirements additional to those of simply constructing a sequence of transformations which will transform a given input n-tuple into a desired output n-tuple: (1) There are lower and upper bounds on the numbers intervening between input and output; thus we might require that no intervening number be less than 0 or greater than 19. (2) Even within these bounds, there can be certain proscriptions; thus we might proscribe any n-tuple containing the number 8 from intervening between start and goal.

A comment is in order concerning our reasons for choosing such an apparently artificial problem domain. Our intention was to design a vehicle with which we could conveniently investigate utilization of theorems or subroutines, one simple enough that most of the program design effort could be concentrated directly on such utilization rather than on matters of other kinds, e.g., pattern recognition. As we shall see, in this we were only partially successful. Further, we wanted problems possessing some of the features of programming or theorem-proving problems but not so many of these features that an extremely complex program would be required to solve any problems at all. Finally, we wanted a problem domain which did not present difficult problems for us. This allows us to take an Olympian view of what our mechanical problem solver is doing.

Let us see how our problems of programming a vector adder compare with programming or theorem-proving problems which people actually have to solve. In our case, there are no difficulties associated with merging two lines of processing into one. Formally this is because the constituent transformations (the basic instructions or rules of inference) are all single-argument functions, a situation which holds neither in real-life programming nor in real-life theorem proving. Further, there are no difficulties in our case associated with branch-

ing or iteration or recursion. Thus our problems are far simpler than actual programming or theorem-proving problems, but they have enough of the features of these actual problems to be interesting. In particular, our problems involve discovering a combination of basic transformations which will effect some particular complex transformation and, further, of discovering an acceptable ordering of these basic transformations. The requirement that they be ordered in a particular way derives from our imposing bounds and proscriptions on an acceptable solution. Thus, in the example given above, if we were to stipulate that no 4-tuple containing a number larger than 10 might intervene between start and goal, the suggested sequence *(S), (T), (T), (W), (W), (W), (W)* would no longer be a solution but the identically constituted though differently ordered sequence *(W), (W), (W), (W), (S), (T), (T)* would be. These various aspects of our problem domain should become clearer with the examples to follow.

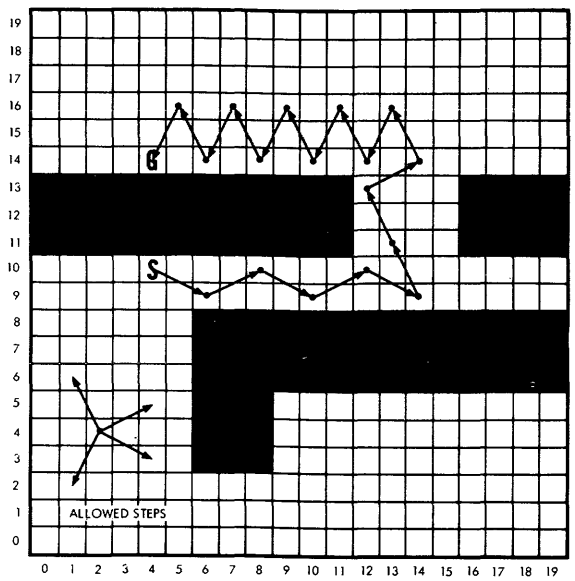


Figure 1. Spatial Interpretation of a 2-Tuple Transformation Problem.

The problem consists in constructing a path, like the one indicated, from S to G. The path must be constituted only of allowed steps and none of these steps can end in a blacked-out cell or out of bounds.

(2.3) *Spatial Interpretation of the Problem Domain*

An Olympian overview is facilitated by giving the n-tuples a spatial interpretation. Thus in the case of pairs (2-tuples), we can correlate the start and goal pairs, the intervening pairs, and the proscribed pairs with cells in a grid. In Figure 1, for instance, we correlate a start pair $<4,10>$ with the cell marked "S," a goal pair $<4,14>$ with the cell marked "G," and the proscribed pairs with the blacked-out cells. Given the pairs $<2,1>$, $<2,-1>$, $<-1,2>$, and $<-1,-2>$ as the basic transformations available, the problem can be interpreted as that of working out a path (like the one in the figure) with these basic transformations as constituent steps, with no step ending in a blacked-out cell, and which begins at S and ends at G.

By supplementing the set of basic transformations of the example so that we have one transformation for each of the eight legal knight's moves in chess, we have the problem of moving a knight around on a chessboard. We have christened the system defined by the four listed transformations the *half-knight calculus* (HKC) and that defined by the full set of eight the *full-knight calculus* (FKC). In order to avail ourselves of the two-dimensional spatial interpretation, most of our work has been with 2-tuple systems like these. (Both the HKC and FKC are, by the way, complete in the sense that their basic transformations are sufficient to transform any pair into any other pair.)

The powers of human visual perception and spatial intuition are very useful in analyzing our mechanical problem solver's performance, in comparing and evaluating its solutions and attempted solutions, and in deciding on procedures which might be useful to it. There is some danger in using this spatial interpretation, however, for one is likely to be misled concerning the nature and difficulty of the problems insofar as the computer is concerned. We would specifically warn the reader against a tendency to look upon the problems as problems of running a maze with the connectivity of the nodes corresponding to the directly perceived connectivity of cells in the grid. And we would suggest that he can grasp the difficulty of solving the problems involved *without* any aid

from visual perception and spatial intuition, as the computer must solve them, if he thinks of what his own situation would be in the five- or six-dimensional case.

(2.4) *A Succession of Problems from Easy to Hard*

Stipulation of lower and upper bounds and of proscribed n-tuples defines a problem field. For our experiments with 2-tuples we defined five fields, each with a lower bound of 0 and an upper bound of 19 but with the differing sets of proscribed pairs represented by the blacked-out cells in Figures 2 through 6. It should be understood, again with respect to possibly misleading aspects of the spatial interpretation, that the research has *not* been concerned with designing a system which would learn local properties of the five fields. Our reasons for restricting the number of fields rather than using a different one for each different problem was solely one of convenience. Since only a part of each field is relevant to a particular problem, the five fields serve to determine a variety of ordering constraints sufficient for our present purposes.

Given the five fields, a succession of 60 progressively more difficult problems was determined by using each of the fields 12 times and for each use defining the numbers for the start pair and the goal pair with a semi-random process controlled so that the difference be-

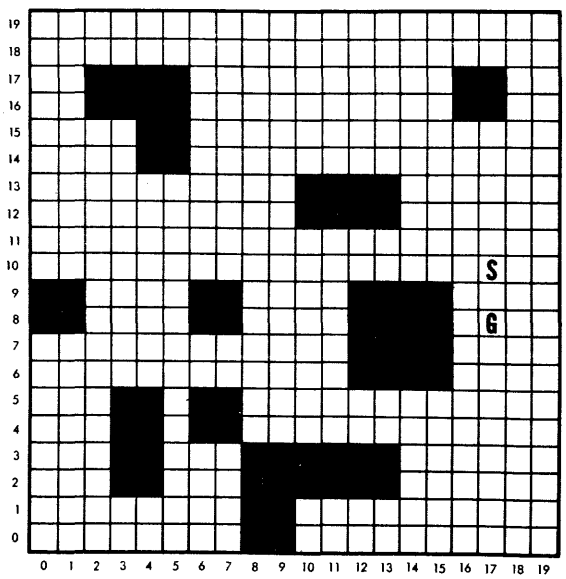


Figure 2. First Problem Field.

The problem illustrated is Problem 6: within the given field of proscribed pairs, transform <17, 10> into <17, 8>.

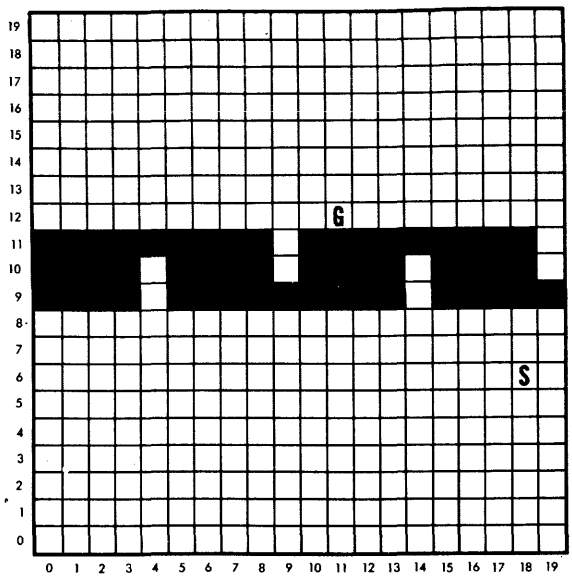


Figure 3. Second Problem Field.

The problem illustrated is Problem 24: within the given field of proscribed pairs, transform <18, 6> into <11, 12>.

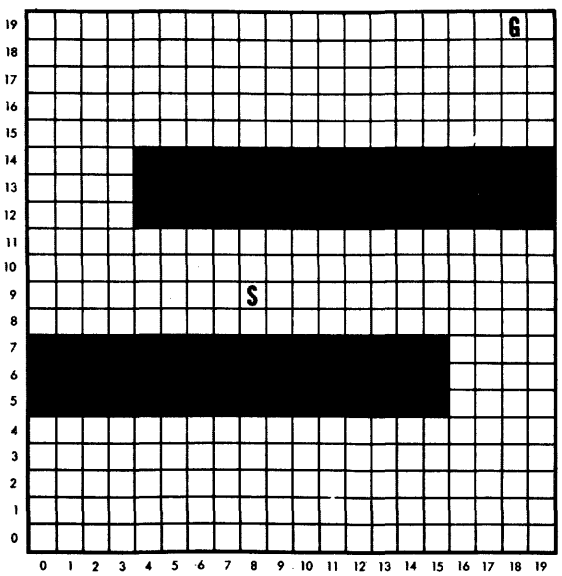


Figure 4. Third Problem Field.

The problem illustrated is Problem 37: within the given field of proscribed pairs, transform <8, 9> into <18, 19>.

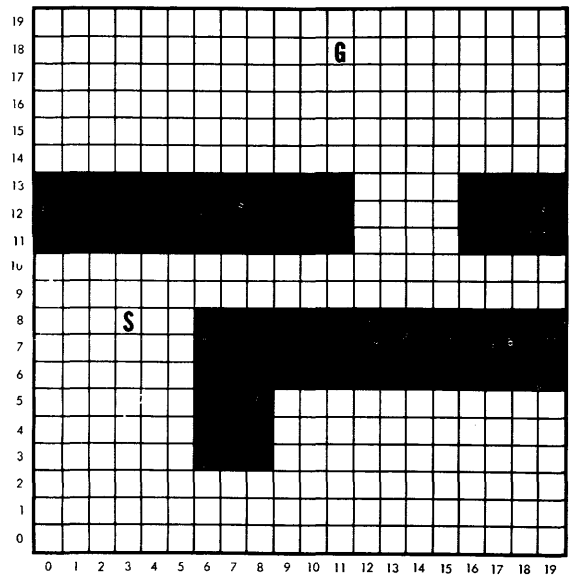


Figure 5. Fourth Problem Field.

The problem illustrated is Problem 48: within the given field of proscribed pairs, transform <3, 8> into <11, 18>.

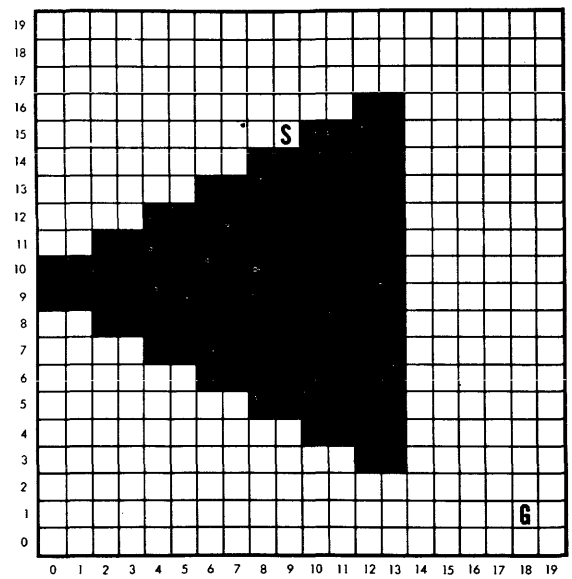


Figure 6. Fifth Problem Field.

The problem illustrated is Problem 55: within the given field of proscribed pairs, transform <9, 15> into <18, 1>.

tween the numbers was likely to be greater for the later problems in the succession. Figures 2 through 6 represent Problems 6, 24, 37, 48, and 55, respectively.

(2.5) *Storage of Theorems*

We can now briefly describe the program with which we are investigating theorem utilization and which so far has been used to test some ideas concerning how utilization of solutions of the easy problems early in the succession of 60 problems might facilitate solution of the more difficult problems later in the succession. The most important idea was that once a problem was solved the solution should be generalized and precisely categorized in a certain way, with the solution's possible relevance to the solution of later, more difficult problems being determined by its categorization. We wished to devise something more efficient than mere addition of new theorems to an unordered theorem list which then had to be scanned in its entirety each time a search was made for possibly useful theorems, as was the case with Newell, Shaw, and Simon's original Logic Theorist (with the exception of one experiment where the theorems were categorized and weighted on the basis of their experienced usefulness in connection with particular methods).

One can use various techniques to make the selection of theorems possibly useful for a particular problem more efficient than a brute-force search of an unordered list. For instance, Newell, Shaw, and Simon used certain similarity tests in the search of the Logic Theorist's theorem list; they experimented with enhancing search efficiency by pre-computing the information for each theorem needed by these tests and storing it along with the theorem, thus obviating its computation each time the list was searched. Although the indexing was inverted, this represents a simple case of the kind of theorem categorization which we have wished to implement and test. Stefferud[23] has modified the Logic Theorist so that previously-proved theorems are stored not on a simple list but as part of a complex list structure (called a theorem map) which compactly represents the syntactical structures of all the theorems available. Theorems are selected for possible usefulness in connection with a given problematic expression by a single operation matching the syntactical structure of the expression against this theorem map. That part of the map with which an appropriate match is obtained determines the whole set of previously-

proved theorems which have the appropriate syntactical structure for use with the given expression. Our approach is not unrelated to Stefferud's but we would like to work out a theorem-selection system which uses clues to the usefulness of a previously-proved theorem more powerful and more selective than whether or not there is a precise match between the syntactical structure of part of the theorem and the syntactical structure of part of the expression to be proved. What such clues might be in the particular case of a propositional calculus like that with which Stefferud is working we leave for consideration in another place; here let us see what we mean for systems like the half-knight and full-knight calculi.

It will be recalled that the problem of constructing a derivation in one of these calculi consists in discovering what combination of basic transformations will transform a start pair into a goal pair and in ordering these transformations so that none of the intervening pairs are out of bounds or on the list of proscriptions. For the experiments here reported what was saved of any particular successful derivation was information about how many applications of each basic transformation were required plus the defined transformation which this combination was able to effect. This was abstracted from the order in which the basic transformations were needed and from the particular start pair and goal pair which had defined the problem. Thus, as illustrated in Figure 7, the sequence $<-\!\!-1,-\!\!-2>$, $<2,1>$, $<-\!\!-1,2>$ might be used to solve the problem of getting from $<1,2>$ to $<1,3>$. What will be saved about this solution is that it uses one each of the indicated basic transformations and that this combination will effect the defined transformation $<0,1>$, i.e., it will transform any pair $<X,Y>$ into $<X,Y+1>$. Now if the system is later confronted with the problem, perhaps as a subproblem of some larger problem, of transforming $<11,8>$ into $<11,9>$ in the problem field also as illustrated in Figure 7, it might retrieve and try to use the information it has saved from the earlier solutions; but after it has retrieved the information it has the additional task of ordering the basic transformations in a manner which satisfies the constraints of the new problem. Because of the

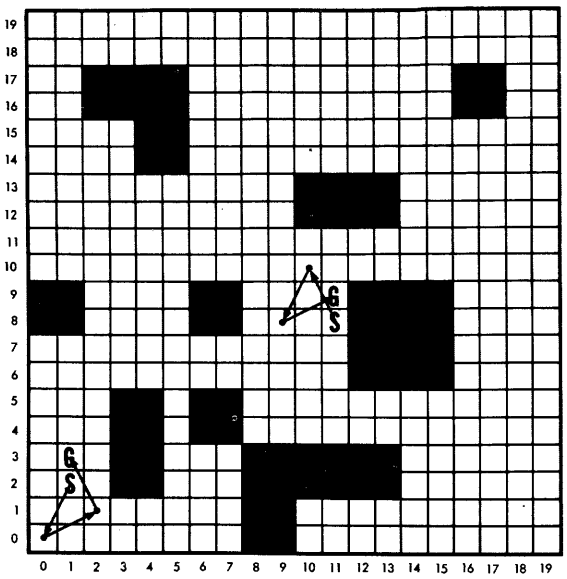


Figure 7. Use of a Previously Developed Solution to Solve a New Problem.

A combination of allowed steps which has been discovered to solve the problem $<1, 2>$ to $<1, 3>$ can later be used to solve $<11, 8>$ to $<11, 9>$. But a different ordering of the steps will be required for an acceptable solution.

role they serve as defined transformations, let us call the abstracted solutions "theorems" even though they may not look much like what the logician usually refers to with this term.

The categories on the basis of which theorems are filed and retrieved must of course be relevant to the information which is saved about the theorems. Four easily computable binary properties have been used to determine these categories. For each theorem let us call the change it makes in the first term of a pair on which it operates its X-value and the change it makes on the second term its Y-value. The four binary properties used to categorize a theorem are whether its X-value is positive, whether its Y-value is positive, whether its absolute X-value is greater than or equal to its absolute Y-value, and whether its absolute Y-value is greater than or equal to its absolute X-value. Tests to determine the values of these properties for a particular theorem are arranged on a simple discrimination tree (which can easily be expanded or otherwise modified for experimentation with alternative proper-

ties). When filed, a theorem is sorted down this three into one of eight categories corresponding to octants in a Cartesian coordinate system. The theorems in one category are put on an ordered list with the position of a theorem on the list being determined by the sum of its absolute X-value and its absolute Y-value. The basic transformations of a calculus are placed on these lists right along with the defined transformations. It will be noticed that the FKC has one basic transformation for each of the eight lists.

(2.6) *Retrieval of Theorems*

For our problem domain, problems can be characterized with exactly the same properties used to characterize theorems. Thus the primary technique used to select a theorem possibly relevant for solution of a problem is to sort the problem down the same discrimination tree used to separate and file theorems. Given the relevant list of theorems, the first selection is that one which leaves the smallest remaining problem. (More precisely: given a problem of transforming S into G, first selected is that theorem T for which the sum of the absolute X-value and absolute Y-value of the problem T(S)-to-G is smallest.) Lower priority selections are made by alternately taking theorems from above and below the first choice on the theorem list being used or, if the list is too short to provide enough selections as determined by an externally specified parameter, by making selections from the lists for neighboring categories.

We should make two clarifying remarks before proceeding with program description. There is a difficult question when specifying a system such as the one being described concerning the criteria which should be used in selecting a theorem to be tried. About the only thing which recommends the size of T(S)-to-G as a primary criterion in our case is that it is easily and quickly computable. A more sophisticated system would attempt an evaluation of the difficulty of actually being able to realize T(S) with T given the ordering constraints; and an evaluation, with measures better than size alone, of the problem T(S)-to-G. Gelernter's oft-cited use of a semantic model was directed toward evaluating the counterpart of T(S)-to-G for his Geometry Theorem Prover. (Working backwards, what his system actually had to evaluate was S-to-T^{-1}(G). It did this by testing whether T^{-1}(G) was true in a model determined by S, the given premises.)

A second clarifying remark concerns sorting problems down the same discrimination tree used to file theorems. In general, there might be properties which would be useful for categorizing theorems but which would be inapplicable to problems or *vice versa*. In our present case, for instance, the total number of basic transformations involved in their definition might be a property useful for categorizing theorems. Problems have no such property. There would be, however, a corresponding problem property whose value could be tested at the same point in the sorting tree. There would have to be such a problem property else the theorem property would not be a useful basis of categorization. It should not be forgotten that the categorization is for purposes of determining relevancy of a theorem to a problem.

To continue with program description: The first thing the program does when attempting to solve problems is to construct a tree of possibly useful theorems. The number of branches from each node is determined by an externally specified parameter N. The program selects N theorems possibly useful for the problem S-to-G; then it recursively selects N theorems for the problem T(S)-to-G determined by the best T of its first N selections; then it recursively selects N theorems for the problem U(T(S))-to-G determined by the best U of its second N selections; etc. Recursion is terminated either when a closure is effected, i.e., when a sequence of selected theorems together transform S into G (ignoring the ordering constraints); or when a maximum allowable depth of recursion, as determined by an externally specified parameter, is reached. At the point when recursion is first terminated, the tree will have the structure indicated in Figure 8. The non-maximum-depth terminal nodes will be extended to full depth only if necessary, i.e., if the theorem choices from the completed part of the tree do not result in a solution.

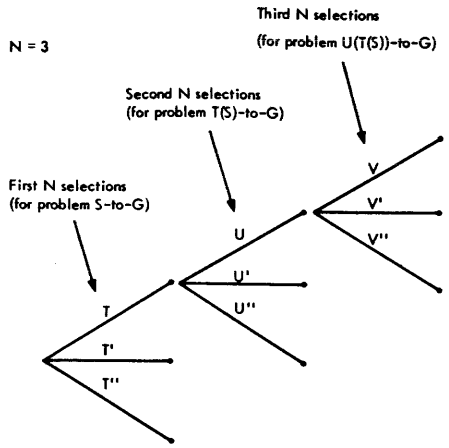


Figure 8. Tree of Theorem Selections.
The tree is extended in the manner indicated until closure is effected or until a recursion limit is reached. If the top line of theorems (T, U. V, etc.) does not result in a solution, other lines (T, U, V', etc.) will be tried. The tree will be extended from the lower nodes, e.g., that for T', when but only when necessary.

(2.7) *Application of Retrieved Theorems*

Given the tree of selected theorems, the program now picks up from the top down (as the tree is represented in Figure 8) sequences of theorems which effect a closure, trying one after another until it attains a solution, runs out of allowed time, or exhausts the tree. Given a closure-effecting sequence of theorems, the program still has a difficult task to accomplish for it must order the constituent basic transformations so that the path of pairs intervening between start and goal is in bounds and does not include any proscribed pairs. The ordering mechanisms constitute the longest and most complex part of the program. (The entire program is approximately 4500 IPL instructions in length.) However, we shall not here describe these mechanisms because their operation is not directly relevant to our present theme of theorem utilization (though, as we shall see, their complexity and the fact that they don't work very well are directly relevant to conclusions to be drawn concerning effective use of theorems by machine). One of these mechanisms uses a recursive technique of breaking problems into subproblems, and we shall say something about this particular, very important mechanism in the next section.

(2.8) *Breaking Problems into Subproblems*

A number of authors, especially Minsky,[11,12] have remarked on the importance of enabling problem-solving programs to transform a large problem into a constellation of smaller problems, "to plan" as Minsky calls it. This has turned out, however, to be a very difficult capability to implement, and there are not many (if any) operating programs which have it, though Newell, Shaw, and Simon[17,18] have done some work toward incorporating a planning capability in their General Problem Solver. We suggest that theorems (or, more generally, what we have called abstracts of previously developed sequences) might well be primary instruments of planning. When planning, a human sequence synthesizer usually is especially interested in discovering a special kind of subproblem, viz., one very like problems he has already solved. A good deal of human sequence-synthesizing activity is an attempt to transform a problem into a constellation of subproblems each of which belongs to a *familiar* kind of problem, familiar in the specific sense of the problem solver's having available abstracts of sequences previously developed as solutions for problems of just this kind. The kinds of familiar problems will be more or less general and the abstracts of previous solutions more or less complete, and that's why, even after retrieval of the abstracts, the familiar problems are still problems. So in our case, the retrieved theorems often won't work. We shall presently want to examine why, but here we simply want to point out the connection between planning and theorem utilization.

There are besides theorem utilization other kinds of planning, many of which appear to be special to particular problem domains. One such special kind is used by our present program when faced with the task of ordering a set of basic steps so that none of them result in a proscribed pair of numbers. When starting on any problem, the program has no way of determining which if any of the given proscriptions in the problem field are relevant to

the problem. The reader can grasp the nature of this difficulty by imagining himself given the proscriptions simply as a list of pairs of numbers rather than in their spatially interpreted form. He then has no easy way of telling which of these pairs might get in the way as he attempts to work out a path between one pair of numbers S and another pair G. The program proceeds, to begin with, as if none of the proscriptions were relevant. As it tries one path after another it compiles a list of those proscribed pairs which have prevented success. This list's reaching a certain size, as determined by an externally specified parameter, is taken to indicate that success is unlikely without taking into consideration the specific configuration of proscriptions which are relevant.

Taking its occurrence on the list as evidence that a particular proscription is relevant, the program determines rows and columns (speaking in terms of the spatial interpretation) of relevant proscriptions and then looks for gaps in these rows and columns through which a successful path is apparently going to have to proceed. Cells within these gaps are then selected as midpoints defining possible subproblems, each such cell M determining the subproblems S-to-M and M-to-G. Several such M's might be selected at one time, and they are put on an ordered list with the position of an M on the list being determined by criteria like the number of proscriptions in its immediate vicinity. The order of this list determines the order in which attempts are made to solve the subproblems determined by the M's.

Given a pair of subproblems S-to-M and M-to-G on which to work, the system attacks these recursively, approaching each with the same mechanisms it originally applied to the problem S-to-G. Thus S-to-M might be broken into S-to-M′ and M′-to-M, and so on. The only difference between operations of the program at one subproblem level and at another is that the lower the level the less the effort allowed per subproblem before reporting failure. The amount of effort expendable is controlled by reducing the potential size of the tree of theorem selections at each recursion onto a subproblem, as illustrated in Figure 9. This reduction also provides an effective means of ensuring that recursion will terminate.

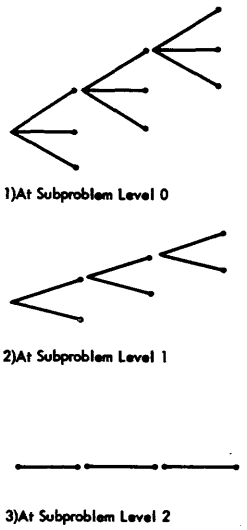


Figure 9. Theorem Selection Trees at Different Subproblem Levels.

The amount of effort spent on subproblems is controlled by reducing the potential number of theorem combinations which can be tried each time descent is made to a lower level.

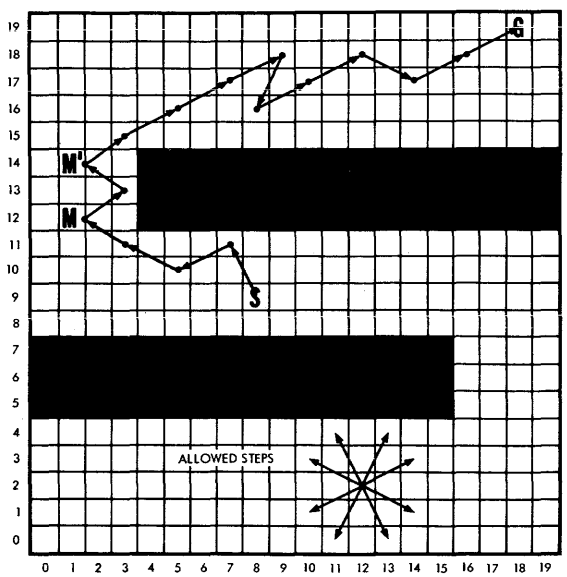


Figure 10. Recursive Solution of Subproblems of Problem 37.

The original problem S-to-G was solved by breaking it into the subproblems S-to-M and M-to-G and recursively attacking these. The problem M-to-G was further broken into M-to-M′ and M′-to-G before success was achieved.

A solution obtained by the method of determining subproblems and recursively attacking them is illustrated in Figure 10. The original problem <8,9> to <18,19> was broken into <8,9> to <1,12> and <1,12> to <18,19>, and the latter of these in turn into <1,12> to <1,14> and <1,14> to <18,19> before success was attained. This simple example does not illustrate the frequent occurrence of a number of alternative midpoints being tried and discarded before usable ones are hit upon.

(2.9) *Deciding Which Theorems to Save*

We earlier raised the problem of improving a theorem file in ways other than simply adding to it. There is built into the program a mechanism for periodically culling from the file theorems which have been of little use. We shall not report more fully on the mechanism in this paper, however, because our experiments have not yet provided us with the opportunity to test the mechanism and to compare it with alternatives.

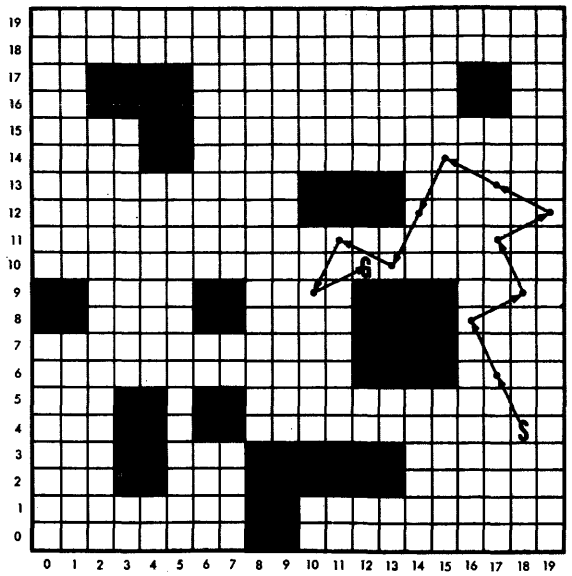


Figure 11. A Good Machine Solution to Problem 56. The program is able to solve problems—in some few cases, elegantly. (The illustrated solution is in the FKC: allowed steps corresponding to the eight legal knight's moves in chess.)

3. RESULTS OF EXPERIMENTS

(3.1) *Vindication of Theorem Utilization*

Sometimes the designer's product doesn't work at all; the airplane crashes or the ship sinks. So the first questions to ask about a complex problem-solving program are "Does the program work when all the pieces are put together? Does it solve problems?" We are happy to report in the present case that the answer is *yes*. The program does solve problems, in some cases quite elegantly (Figure 11 shows one machine solution of Problem 56 in the FKC), in other cases very inelegantly (Figure 12 shows another machine solution of the same problem), but of course in some cases not all (Figure 13 shows a good but unsuccessful try for Problem 38 in FKC).

More to the specific point of the experiments, the problem-solving power of the system was substantially increased by its ability to utilize theorems. We tested this by comparing its performance on Problems 51 through 60 without any transformations in its file except the basic ones of the FKC and its performance on these same ten problems with its file containing the basic transformations of the FKC plus all the defined transformations (about 60) which it had accumulated from solving those which it could of Problems 1 through 50 and associated subproblems. Almost all of the time of the theo-

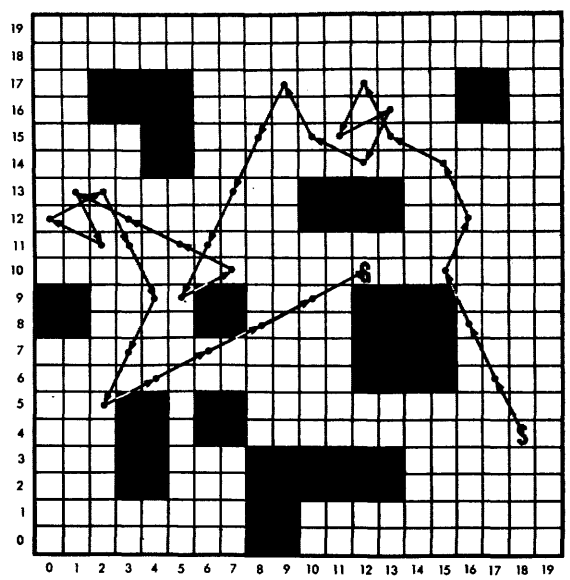


Figure 12. A Poor Machine Solution to Problem 56. The program is able to solve problems—but, in many cases, very inelegantly. (The illustrated solution is in the FKC.)

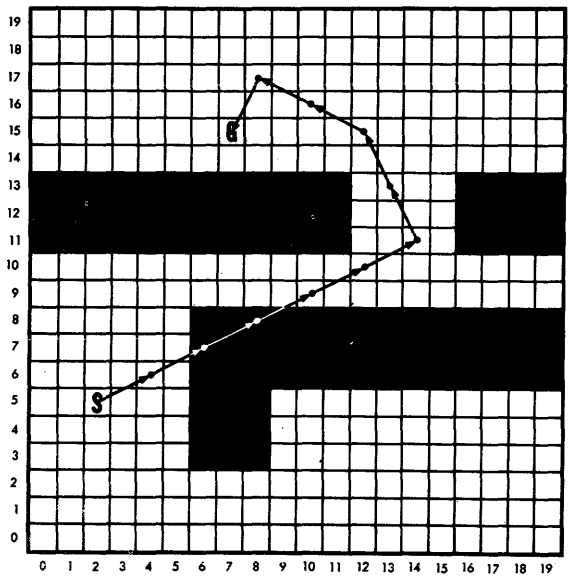


Figure 13. A Good But Unsuccessful Machine Try.
The program in any of its versions was unable to solve Problem 38. To get an idea of the kind of ordering constraints the program is up against, the reader might try moving a knight from S to G without letting it touch down on any of the blacked-out cells.

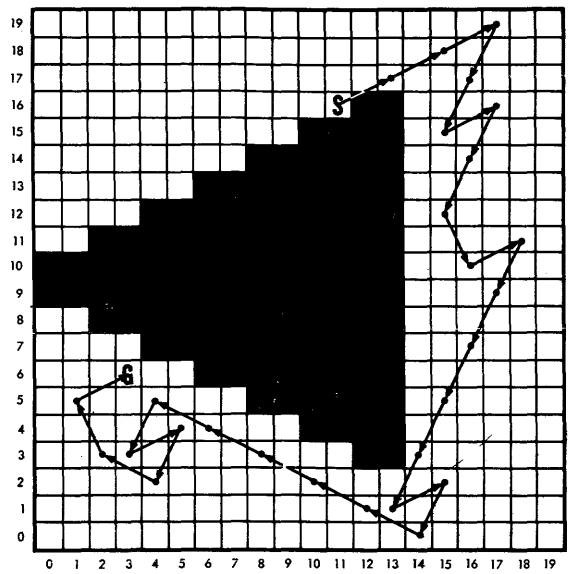


Figure 14. A Hard-Won Machine Solution of Problem 60.
Even with the use of previously-provided theorems, the achievement of this solution required a great deal of effort (over a million IPL cycles). Without theorem utilization the program was unable to solve the problem at all. (The illustrated solution is in the FKC.)

remless machine was spent in searching for combinations of the basic transformations which together would transform start pair into goal pair. Given its difficulty in discovering any combination at all which would effect this closure, it was unable to solve any but the simplest of the ten problems. Successful solution of the more difficult problems required having a variety of these combinations easily available from which to discover one which could be ordered to satisfy the ordering constraints. The machine with theorems was quickly able to discover combinations of basic transformations which effected a closure and could spend its time attempting to arrange these combinations in an acceptable order, quickly discovering an alternative combination in case further work on some given combination appeared futile.

Examination of the relative performance of the two machines (the theoremless machine and the machine with theorems) on Problem 60 is instructive. Figure 14 illustrates the solution which the machine with theorems was able to achieve in 48.5 minutes (and in 1,276,809 IPL cycles). This solution involved a total of 13 different closures. In contrast, the theoremless machine in 60 minutes effected only two closures, with combinations which it was unable to order satisfactorily, and fruitlessly spent the rest of the hour allowed it trying to effect a third.

In at least one important sense the HKC and FKC derivations required in the present experiments are not trivial; this is with respect to the very large number of basic transformations required to effect them. For the problem illustrated in Figure 15, for instance, the solution achieved by our program (with theorems) is 44 steps long; an apparently optimum solution (achieved by a human being) is 28 steps long. A question raised by these performances is whether there are any general strategies (heuristics), other than utilization of previously developed subsequences, powerful enough to enable synthesis of very long sequences in non-trivial cases like theorem proving and programming. It might be suggested that all the machine needs is something like the space-perception ability which enabled the human

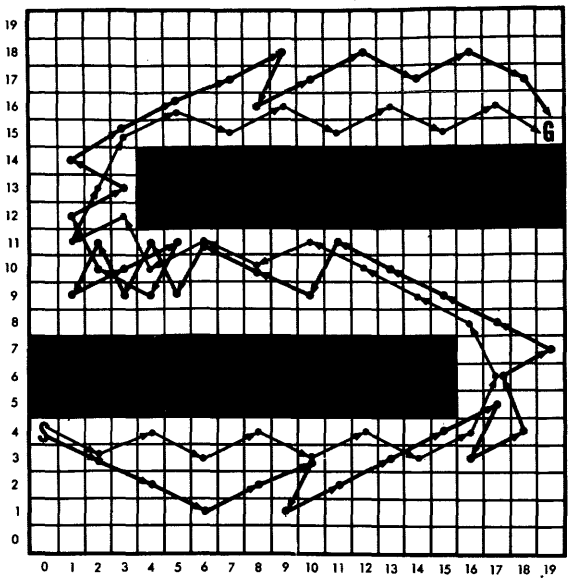


Figure 15. Machine and Human Syntheses of Long Sequences.

The machine-produced sequence (heavy lines) is 44 steps long, the human-produced sequence 28 steps long. The human's visual abilities enabled him to get along without much prior experience on this kind of task. But without such abilities the machine needed prior experience to synthesize a sequence so long. (The illustrated solutions are in the FKC.)

being to synthesize his 28-step sequence without much prior experience on this kind of task. But this is just to shift the problem to another domain. Given the spatial interpretation of the problem, the human being makes extensive use of experience, if not ontogenetic then phylogenetic.

So the question still stands. Certainly no mechanical game player which looks ahead move by move is able to come close to looking 28 (or 44) moves ahead. Samuel's checker player[20] might achieve such farsightedness but it is of interest to note that this is with a technique which is a kind of utilization of previously-developed (trees of) sequences. Whenever the checker player, using what Samuel calls its rote memory, applies to positions X moves ahead a previously-computed evaluation which involved looking Y moves ahead, this provides it with an evaluation identical with what it would achieve by directly synthesizing and evaluating sequences X+Y moves in length. The effect is cumulative so that eventually the checker player is able to look ahead a very great distance. Samuel's work, by the way, leads to an important conclusion relative to what we have called the theorem-utilization problem: however powerful theorem utilization might be, attainable degree of theorem generalization and attainable efficiency of theorem retrieval impose limits. Samuel was able to use retrieval techniques as efficient as binary search but was unable to achieve any significant degree of generalization; as a consequence his file of experience would reach an unmanageable size before it enabled his machine to play good checkers at any stage of the game. He had to turn to techniques other than "theorem utilization" in order to achieve good mid-game playing ability. It should be noted, however, that these other techniques were not alternative ways of synthesizing longer sequences but were improved ways of evaluating given sequences of limited length. Hence, our question *still* stands: are there any strategies, other than utilization of previously-developed sequences, powerful enough to enable synthesizing of very long sequences in difficult problem domains if this is what we have to do, say to program a computer?

We might apply the question directly to Newell, Shaw, and Simon's General Problem Solver (GPS).[17] The primitive operators used by GPS correspond to what we have called basic transformations of a sequence-synthesizing problem domain. Given this correspondence, the many similarities between the operation of our program in its theoremless mode and the operation of GPS are obvious. Of course, adding to GPS an ability to accumulate and efficiently to use defined operators may not prove especially difficult, though it raises questions of exactly the kind with which we are concerned in this paper.

To complete our case for the indispensability of theorem utilization for difficult sequence synthesis, we might temporarily look up from our experiments and appeal to authority. After making the point that non-trivial logical deductions usually require many steps of inference, Carnap[2] (pp. 36-37) remarks, "In practice a deduction in science is usually made by a few jumps instead of many steps. It would, of

course, be practically impossible to give each deduction which occurs the form of a complete derivation in the logical calculus, i.e., to dissolve it into single steps of such a kind that each step is the application of one of the rules of transformation of the calculus. . . . An ordinary reasoning of a few seconds would then take days." We suggest that, at least in this case, as it goes for men, so it goes for machines.

(3.2) *Failure to Satisfy Goal Criterion*

We must now turn to reporting and analyzing less positive results of our experiments. Over-all, our program was not a very good problem solver. For the HKC, in two runs (differing in respects inessential to our present interest) through the entire succession of 60 problems, the program was unable to solve 33 in one case and 30 in the other. For the FKC, in two runs the numbers were 21 and 17. Of course, increasing the time allowed per problem would have improved this performance somewhat, but we experimented enough with this variable safely to conclude that the time limit would have had to be extended unreasonably to assure success with most of the unsolved problems and, in any case, time taken to achieve solution is an important measure of the power of a heuristic problem solver. Even more disappointing than these tallies of unsolved problems was their distribution. It was mainly problems in the later part of the succession of 60 which were not solved, the difficult problems requiring for their solution use of solutions obtained for problems occurring earlier in the succession. The obvious conclusion is that "theorem utilization" wasn't working very well.

At first we thought that perhaps the failure was a matter of the succession of problems not being long enough, i.e., that the program just needed more simple problems from which to build up a store of experience before being able successfully to attack the difficult ones. We attempted a minor empirical test of this possibility by giving the program a second chance on the unsolved problems from the first run through the succession of 60, a second chance during which it could use for any given problem theorems acquired after its first attempt on the problem in addition to the theorems which had been available at the time of the first attempt. No more of the difficult problems were solved with the additional theorems than had been solved in the first place.

(3.3) *Implications of the Failure for Theorem Utilization*

In the difficult cases the program was failing because it did not possess techniques powerful enough to deal with severe ordering constraints. For problems like Problem 48 illustrated in Figure 5, for instance, even when the program had discovered a usable combination of basic steps, ordering them in the precise way necessary to avoid the proscribed cells required something the program does not possess, viz., a sharp sensitivity to patterns formed by these cells. The best the program could do on Problem 48 is illustrated in Figure 16.

Of course, the sharp sensitivity alone would be of little value. There would be no point to the program's being able to recognize a pattern of proscriptions if this didn't indicate

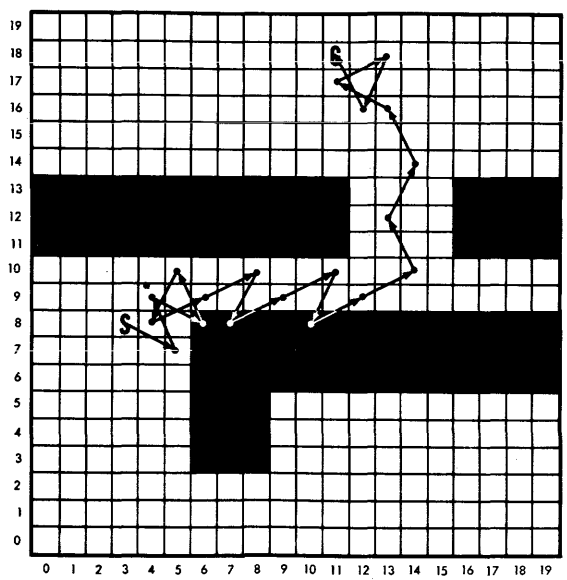


Figure 16. Machine Inability to Achieve the Precise Ordering Required to Solve Problem 48.

Appearances notwithstanding, the program is a long way from attaining an ordering of steps which will move it through the tight part of the illustrated problem field. Most of the program's failures are attributable to weaknesses of its ordering procedures—despite the fact that these constitute the bulk of the program. (The illustrated attempt is in the FKC.)

something for it to do. Recognition of a narrow channel through which the successful path must proceed would have to suggest procedures for selecting the particular basic steps and an ordering of them which would effect a path of acceptable dimensions; similarly for recognition of a required sharp right bend, or of a required detour to the left, or of a circumscribing rectangle within a certain size range and with entrance and exit required at certain places, etc. So the system would need a much more powerful pattern-recognition capability than it presently has; but it would also need a system connecting this capability with methods of behaving appropriately given certain patterns.

It will be recalled that information about the order of basic transformations in a successful sequence is discarded by the present program when that sequence is abstracted and stored as a previously-proved "theorem." There is no point to saving information about order when the pattern-recognition system used for retrieval of theorems relevant to the solution of particular problems is able to recognize only direction and distance aspects of the problems and nothing about ordering constraints. However, given the appropriate pattern-recognition capability there would be reason for saving information about order. For our particular problem domain (as well as for many others), one of the things which makes a problem difficult is its ordering constraints. Because our system is unable to recognize clues involving these constraints, and with these clues to retrieve relevant information from its file of past experience, it falls down badly on problems involving severe ordering constraints. We need, then, two complementary things: improved pattern-recognition capability and different kinds of information saved about our previously-proved theorems.

It might be contended that we are begging the question. Surely there are ways to satisfy severe ordering contraints other than the retrieval and use of orderings previously worked out. To this charge we reply that there might well be such alternative ways but there appear to be sharp limits on how far one can go with them. This is related to the point we made earlier about the apparent necessity of theorem utilization for the mechanical synthesis of non-trivial, long sequences. One can, of course, write a more and more complex program, but at some point he runs out of programming time or out of program storage space in his computer. The limits we are talking about are practical. Also, a point of special importance for artificial intelligence research (see Kelly and Selfridge[8]), there is a serious question whether the more and more complex program would not necessarily have to be more and more *ad hoc* with respect to a particular problem domain. These possibly rash statements are at least partly based on our experience with the present program.

We have tried a long list of techniques intended to increase the ordering ability of our program. Our hope has been to achieve a program of sufficient problem-solving capability that we could use it as a vehicle for experimenting with alternate schemes for accumulating, culling, abstracting, filing, retrieving, and applying theorems; and we did not want the ordering aspects of the problems in our problem domain to enter into these investigations concerning theorems because of the very great difficulty of programming the pattern-recognition mechanisms which would be required. Hence, our valiant try to get the program to handle the ordering aspects of its problems by means other than theorem utilization. The try has failed, however, and so we have not so far been able to experiment with alternate schemes of theorem utilization; the reason is that changes in performance due to changes in these schemes are masked by effects due to weaknesses in parts of the program which were supposed to perform their functions without theorem utilization.

We salvage something from the failure if we learn a lesson concerning the importance of theorem utilization even for those parts of our problems where we thought we could get along without it. The reader will be helped in deciding whether he agrees with us that we can't get along without it, if we indicate some of the things we did in trying to do so. We have already described probably the most important thing, viz., having our program look for gaps through which a successful path apparently will have to proceed and use these gaps as bases for breaking a given problem into subproblems.

Several other things were tried, many of them rather difficult to implement. Actually the reason we conducted as many as four separate runs through the succession of 60 problems was that with each successive run we hoped (to no avail) that we had devised significantly improved ordering techniques. We tried a technique for shortening paths, and thereby reducing the likelihood of violating proscriptions, by replacing parts of unacceptable paths with theorems spanning the same gaps but involving fewer basic steps. This caused over-all performance to deteriorate; the program did better by considering many alternative paths, rather than considering only a few and spending a lot of time on each of these trying to edit it into acceptable form. We tried techniques so easy as blindly reversing the order of steps in a path and so difficult as determining and effecting permutations which would usually move a bothersome part of a path from an area containing proscribed cells into an area containing none or at least fewer. We programmed and tried a complex process intended to relax the ordering constraints on a problem by discovering pairs, alternative to the given start and goal pairs, into which the given pairs could easily be transformed but lying in an area allowing greater freedom of choice. We programmed and tried a complex process which packed the steps of a path into an area of minimum dimensions. None of these things worked very well. The simple fact of the matter is we have been unable to devise and realize effective ordering techniques which make no use of previous experience on simpler problems.

It might be suggested that we have boxed ourselves in with the kind of theorem utilization that we do have, that rather than selecting theorems which determine the constituent steps of a path and then being faced with ordering and re-ordering these steps, a problem solver in our domain should select basic steps one by one, selecting a step only if this can be combined with previously selected steps in a manner that satisfies ordering constraints. It seems to us that the experiment reported above with our program in theoremless mode is relevant to an answer. In this mode the program has no alternative but to select basic steps one by one, and it will be recalled what a difficult time it has merely selecting a long sequence of steps which will effect a closure let alone a sequence which satisfies severe ordering constraints as well.

4. GENERAL CONCLUSIONS

(4.1) *The Significance of Theorem Utilization (Again)*

We want now to indicate the significance we find in our experiments for matters beyond better ways of programming a computer to move a knight around on a chessboard. First, let us repeat—for emphasis—our main point concerning the apparent importance of using previously-developed, defined transformations for tasks involving synthesis of long, order-constrained sequences of certain given basic transformations. That the ability to save and use previous experience in this way is important for a man is apparent enough to anyone who will carefully observe himself proving a theorem or writing a computer program. Perhaps we have gone some way toward determining just how important such an ability might be for a problem-solving automaton as well. But we have said enough about this.

(4.2) *Importance of Pattern Recognition for Problem Solving*

Second, we would remark on the importance of pattern-recognition capabilities to effective theorem utilization and hence to effective problem solving. The conclusion to which we are led is that a bottleneck presently preventing more powerful problem-solving machines is perceptual impotence. Our machines won't be good problem solvers until they are good theorem utilizers, and they won't be good theorem utilizers until they are good pattern recognizers. Hard as this bottleneck might be to break, unfortunately it isn't the only thing inhibiting progress.

(4.3) *Program Organization Difficulties*

Third, we would say something about some thorny problems of program organization that arise in connection with a complex heuristic program. We have discovered as have other experimental programmers (e.g., cf. Newell[13]) that a problem-solving program which breaks

problems into subproblems and recursively attacks these, breaks the subproblems into subsubproblems and recursively attacks these, etc., will likely be a rather poor problem solver unless there are some rather strict controls imposed on its recursion. Such a program is likely to dig itself into a hole and never get up and out of the hole to a position from which it can recognize that it should have been digging in a different location in the first place. Perhaps even worse it is not able to relate the different subproblems which it defines at different levels of recursion to each other to discover overlap, duplication, and the possible significance of joint solution of subproblems quite separated from each other on the subproblem tree. And, finally, when it fails at some point and has to decide what subproblem to turn to next, obviously it would be a better problem solver if it were able to base its decision on some kind of overview of the subproblem tree rather than turning automatically to the next subproblem one level up from the point in the tree at which it failed, but this is precluded in the case of uncontrolled recursion. We can report that all these difficulties (and more) arose in our case, and we have no general resolution of them to offer. However, it is perhaps worth reporting that we found very useful, indeed even indispensable, the technique of building up a tree of subproblem records (isomorphic with the tree of subproblems solved and unsolved, pending and disposed of) which could be consulted by the program at any stage or level of recursion when deciding whether to create a new subproblem or whether to go to work on some previously created subproblem. Even though the techniques used for examining and comparing the records on this tree were quick and dirty, controlling recursion with the tree served to prevent much duplication of effort and misinterpretation of the relative importance of particular subproblems.

(4.4) *Harder Problems (for Programs and for Programmers)*

Finally, something should be said about where we see the main difficulties if we are to implement an experience-utilizing system of the kind here discussed for proving theorems in a calculus of greater complexity and sophistication than the HKC or FKC, say for the propositional calculus or the predicate calculus, or for programming computers more powerful and interesting than vector adders. The first problem will be in determining a useful set of properties with which to categorize theorems or subroutines and a corresponding set with which to categorize problems and then, of course, developing pattern-recognition procedures which enable detection of these properties. The second, and it will be recalled from our discussion above that we consider this a problem complementary to the categorization problem, will be determining how successful solutions should be abstracted, i.e., what kinds of information about them ought to be filed, and then, of course, developing procedures which can perform the desired abstraction. But these two problems are overshadowed by one of a kind which has not yet been solved, by us or by anybody else, even for systems so simple as the HKC and the FKC. This is the problem of generalizing a number of particular, previously-developed solutions into a single solution schema.

We haven't said anything about this generalization so far because our present program doesn't do it, instead saving and using only particular solutions. But skilled human problem solvers obviously aren't so limited in their use of previous experience and much of their power derives from their ability to use general schemata which have been built from many separate, particular solutions. We can easily illustrate the kind of general schemata we are talking about by looking at the HKC. Figure 17 shows a human solution of Problem 58 in the HKC. (Problem 58 was much too difficult for our program, in any of its several different versions, to solve.) Several parts of the successful path might have been constructed by using general schemata (and were so constructed, if we are to believe the introspection of the human who solved the problem.) Thus in synthesizing the lower part of the path manifesting a regular alternation of the basic steps <—1,—2> and <—1,2>, the problem solver could have put to good use a general schema enabling left movement through any required distance within minimum vertical dimensions simply by alternately applying

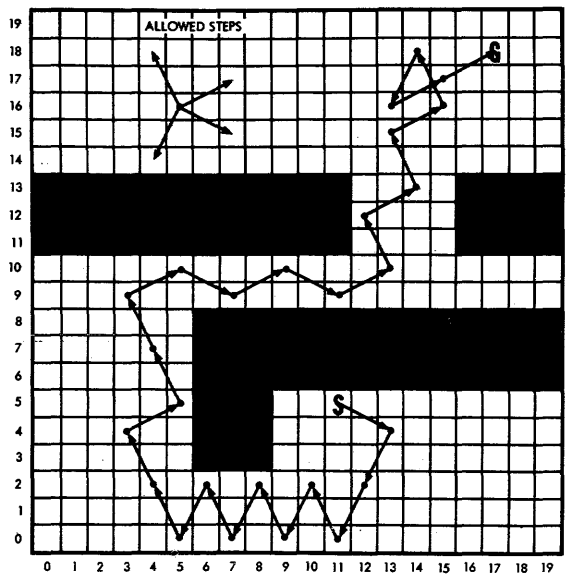


Figure 17. Human Solution (of Problem 58) Requiring Extremely Complex Ordering.

The human problem solver used general schemata. Thus rather than moving from <12, 2> to <4, 2> by remembering and applying some combination of steps which effects a specific <—8, 0> transformation, he used a general schema for moving left any distance required. (The illustrated solution is in the HKC; notice the allowed steps.)

these two basic steps. As it now stands, the program saves from its experience much more specific information, viz., information about how to effect the left-moving transformations <—2,0>, <—4,0>, <—3,2>, <—5,2>, <—3,—2>, <—5,—2>, etc. But it can not generalize this specific information into the schema enabling left movement through any desired distance. A human being apparently has no difficulty making such a generalization. In fact, even rats don't have any difficulty in making a generalization as simple as the one illustrated (cf. Krechevsky).[10] But we are surely up against one of the *hard* problems of computer learning.

For the present we are concentrating on methods of representing general schemata within a computer and methods of using them to solve problems. To take an elementary case, in the propositional calculus we are faced not with representing and using as independent, isolated items the particular theorems $(P \rightarrow Q) \rightarrow (\sim Q \rightarrow \sim P)$, $(P \rightarrow \sim Q) \rightarrow (Q \rightarrow \sim P)$, etc., but with representing and using the general transposition schema under which are subsumed all four theorems with which one can transpose a conditional. The problems of machine representation and use are hard enough. Maybe their solution will suggest some ideas about how to solve the more interesting problem of getting a machine to generalize and establish such schemata by itself, or at least with no more help than a human student has. But these problems of machine learning of general concepts are so hard as to suggest that maybe we ought to relinquish this faith that artifacts can be made to think. Maybe we ought to become Platonists all.

ACKNOWLEDGEMENTS

The research described is part of the independent research program of System Development Corporation. The particular computer program described was conceived while I was participating in the Research Institute in Heuristic Programming held at RAND during the summer of 1962. I wish to acknowledge my indebtedness to Institute staff members for their guidance during the initial stages of writing and debugging the program. Also I want to thank J. Feldman, T. Ruggles, and S. Shaffer for helping me construct the list-processing system required for efficient running of the program on a Philco 2000. Special credit is due Dr. Feldman as designer of the initial version of the system (see Feldman[4]). The system in its present form is a one-address variant of IPL-V (see Newell *et al.*[14]).

REFERENCES

1. AMAREL, S. On the automatic formation of a computer program which represents a theory. *In* Self-organizing system, 1962, edited by M. Yovits *et al.* Washington: Spartan, 1962. Pp. 107-175.
2. CARNAP, R. Foundations of logic and mathematics. Chicago: University of Chicago Press, 1939.
3. DAVIS, M., and PUTNAM, H. A computing procedure for quantification theory. *Journal of the ACM.* 7(1960) :201-215.
4. FELDMAN, J. TALL—A list processor for the Philco 2000 computer. *Communications of the ACM.* 5(1962) :484-485.

5. GELERNTER, H. Realization of a geometry theorem-proving machine. *In* Information processing. Paris: UNESCO, 1959. Pp. 273-282.

6. GELERNTER, H., *et al.* Empirical explorations of the geometry theorem machine. *Proceedings of the 1960 Western Joint Computer Conference.* Pp. 143-147.

7. GILMORE, P. A proof method for quantification theory. *IBM Journal of Research and Development.* 4(1960) :28-35.

8. KELLY, J., and SELFRIDGE, O. Sophistication in computers: a disagreement. *IRE Transactions on Information Theory.* IT-8 (1962) :78-80.

9. KILBURN, T., GRIMSDALE, R., and SUMMER, F. Experiments in machine learning and thinking. *In* Information processing. Paris: UNESCO, 1959. Pp. 303-309.

10. KRECHEVSKY, I. "Hypotheses" in rats. *Psychological Review.* 39(1932) :516-532.

11. MINSKY, M. Heuristic aspects of the artificial intelligence problem. Lincoln Laboratory Group Report 34-55, 1956.

12. MINSKY, M. Steps toward artificial intelligence. *Proceedings of the IRE.* 49 (1961) :8-30.

13. NEWELL, A. Some problems of basic organization in problem-solving programs. *In* Self-organizing systems, 1962, edited by M. Yovits *et al.* Washington: Spartan, 1962. Pp. 393-423.

14. NEWELL, A., *et al.* Information Processing Language—V Manual. Englewood Cliffs: Prentice-Hall, 1961.

15. NEWELL, A., and SIMON, H. The logic theory machine. *IRE Transactions on Information Theory.* IT-2(1956) :61-79.

16. NEWELL, A., SHAW, J., and SIMON, H. Empirical explorations of the logic theory machine. *Proceedings of the 1957 Western Joint Computer Conference.* Pp. 218-239.

17. NEWELL, A., SHAW, J., and SIMON, H. Report on a general problem-solving program. *In* Information processing. Paris: UNESCO, 1959. Pp. 256-264.

18. NEWELL, A., SHAW, J., and SIMON, H. The processes of creative thinking. *In* Contemporary approaches to creative thinking, edited by H. Gruber *et al.* New York: Atherton, 1962. Pp. 63-119.

19. ROBINSON, J. Theorem-proving on the computer. *Journal of the ACM.* 10(1963): 163-174.

20. SAMUEL, A. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development.* 3(1959) :210-229.

21. SIMON, H. Experiments with the heuristic compiler. *Journal of the ACM.* 10(1963): 493-506.

22. SLAGLE, J. A heuristic program that solves symbolic integration problems in freshman calculus. *Journal of the ACM.* 10 (1963) :507-520.

23. STEFFERUD, E. The logic theory machine: a model heuristic program. RAND Memorandum RM-3731-CC, 1963.

24. WANG, H. Toward mechanical mathematics. *In* A survey of mathematical logic, by H. Wang. Amsterdam: North-Holland, 1963. Pp. 224-268.

ANALYTICAL TECHNIQUE FOR AUTOMATIC DATA PROCESSING EQUIPMENT ACQUISITION

Solomon Rosenthal
Directorate of Data Automation
Headquarters, U.S. Air Force
Washington 25, D. C.

In this paper we describe in considerable detail the "Analytical Technique for Automatic Data Processing Equipment Acquisition" developed and documented with Mr. Ernest L. Holt and Mr. Joseph T. Averitt for the Directorate of Data Automation, Headquarters, U.S. Air Force. This is the first public description of this Technique and how to use it.

The Technique is designed to facilitate competitive selection of Electronic Data Processing Equipment regardless of the purpose for which the equipment is to be acquired. It is equally applicable for centralized or pick-your-own concepts of selection.

There are four parts, each designed as part of an integrated whole. The parts are:

I Instructions for preparation of specifications for submission to vendors.

II Instructions for submission of EDPS proposals.

III Instructions for validation of EDPS proposals.

IV Instructions for scoring of EDPS proposals.

Detailed knowledge of the other three parts is not required to implement any one part, but to insure effective use of the Technique, those charged with the responsibility for its overall administration must be thoroughly familiar with all four parts.

This system, including a weighted factor selection method, was designed to be flexible, impartial, efficient and effective. Flexibility is achieved by permitting factor and weight modifications which best suit each individual selection. To maintain impartiality the factors considered and the weights assigned are varied prior to solicitation of proposals. The development of the technique was not influenced by applications to be processed on selected equipment, nor by specific equipment characteristics or considerations for any vendor. Equipment selection time is shortened, the resulting selection is valid, and the documentation produced fully supports the selection. Due to the built-in flexibility, impartiality and efficiency of the technique, it is acceptable to the ultimate user of the selected equipment, to approval authorities, and to vendors—in short, it is effective!

PART I

The "Specifications Module," or "Specs for Specs" as some like to refer to it, was developed to provide a standard arrangement, organization and approach to the problem of preparation of the EDPS specifications for submission to vendors. A more complete description would provide a table of contents for specifications, and detailed instructions for the contents of each chapter and paragraph. Here, rather than present the prescribed format, we will discuss the material to be included in the specifications.

Specifications prepared in accordance with the standard format provide sufficient information about the job to be done to permit interested vendors to submit proposals that are truly responsive, and also to establish for the scorer a firm basis for application of the scoring module to be described later.

Prospective bidders must be fully informed about intended locations and the user's organizational mission and objectives. The user's plans for conversion from any existing system; for centralized or decentralized programming in the case of multiple installations; his equipment delivery and implementation schedule; and his intention to pilot test before ordering more than one system, are all bits of information a vendor needs to know before investing in a proposal.

Any conditions or minimum standards, with which the vendors must comply, must be documented. These specific requirements, or restrictive criteria, should be held to a minimum, and must not be designed to eliminate any vendors from consideration. If these rules are followed, it seems reasonable to state that non-compliance with any of these specifics could result in a proposal being eliminated from competition. Restrictions might justifiably apply in such areas as:

Delivery (Dates and Places)

Cost (Maximum Funds Available)

Timeliness (Allowable delay between remote station inquiry and response, etc.)

Support (Programming, maintenance, systems analysis, etc.)

Physical Environment (Room size, controls, floor)

Equipment Characteristics (Print positions, character sets, random access storage, etc.)

Compatibility (With data banks, programs, other interface requirements)

Expansion Capability (Anticipated workloads)

Software (Monitors, executives, scientific subroutines, common languages, etc.)

Others

We must remember that to qualify for inclusion in the above list an item must be a firm, unalterable requirement.

The heart of a specifications package is that part which documents the job to be done. For each data processing task, a statement is required which fully describes a representative sample of the total workload of that type. As large a percentage of the total as is practicable must be stated for each task. A sufficient number of task statements must be made to indicate collectively the requirement for all computer characteristics involved, i.e., an adequate number to enable determination of the configuration and capacities needed to accomplish the tasks.

A complete statement includes a general description of the job and the functional areas involved (personnel, materiel, accounting, scientific research, etc.). All processing requirements must be described in narrative form. All items, documents, files, master files, tables, etc., which are to be used during—or produced by each processing step, should be discussed. Sort runs, edits, and so forth should be explained. Flow charts, consecutively numbered, describing the data system must be prepared. Each processing step must be identified and numbered to correspond to numbered paragraphs in the narrative. All inputs to, and outputs from, every chart must be clearly labeled to show their source or destination and to tie all the charts together. All inputs and outputs must be described in terms of the number of Alpha and Numeric characters contained. If sample document forms are available they should be included.

A key item in our system is the "Workload Statement Chart" which must be supplied with the specifications; it provides a tabular summary of the information contained in the job statement just covered. A separate page or chart should be produced for each application or task. A more complete description would leave nothing to the imagination in the instructions for completing this chart. Here, we merely present a recommended "Workload Statement Chart" (Figure 1). Just one word of caution; the entries must serve to explain what came before, not to confuse the reader. In other words it should be simple for anyone to relate each entry to a corresponding part of the narrative and applicable flow chart.

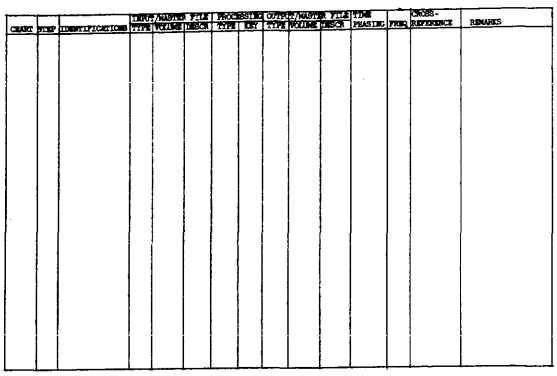


Figure 1. Workload Statement Chart.

The final entry on the "Workload Statement Chart" for each task should be "The above statement represents *'X'*% of the total workload of this type." The final entry on the last chart should be "The above statement(s) represent(s) *'X'*% of the known total workload." The percentages indicated will be used later in proposal preparation to project an estimate of the total time each proposed equipment will require to accomplish the total requirements.

In most cases it will be advantageous to give prospective bidders a description of the intended site(s). Include such information as available floor space, loading factors, maximum distance allowable beteen main computer and remotes, etc. To enable the vendors to propose a complete system, including power converters, regulators and other items needed, tell them what kind of facilities exist. You might also wish to tell them your current inventory of such things as magnetic tapes and control panels to give them a basis for quoting some trade-in values.

PART II

The "Proposal Module" serves a dual purpose. First, it provides those requesting a proposal with a complete guide to designing that request. Second, it provides the vendors with a "recommended" standard arrangement and indexing of the many subjects to be covered in a proposal. Therefore this module should have two prefaces —one to be read by those preparing the request, the other to accompany the request, to be read by those responding with proposals.

These prefaces should include,

In the First:

This part is used in *preparation of instructions to vendors.* These instructions must be tailored to the requirements of the particular selection by those forwarding specifications and requests for proposals to vendors.

In the Second:

We request that your proposal *be prepared as outlined* in the attached "instructions for proposal organization."

* * * * * * * * * * * * *

This format is not intended to restrict the contents of your proposal. The entire contents of Chapters I thru IX must relate to the specific configuration of equipment on which your proposal is based.

Your proposal should include only that one equipment configuration adjudged by you to be the best for the purpose of this selection. If you consider it necessary to propose more than one configuration, each must be submitted as a separate and complete proposal (Chapters I thru IX).

Following Chapter IX, you may furnish any additional chapters which you consider pertinent to your proposal.

When proposals are prepared in a standard format a giant step has already been taken towards simplifying the comparison of proposals. It will be possible to pull from each proposal a chapter or paragraph, a prescribed chart or table, and compare the proverbial apples with apples, not with oranges or mangoes.

The standard format in which proposals should be submitted, and the kind of information to be included, is:

Chapter I General

A letter of transmittal and a brief summary, for executive review, of the highlights of the proposal.

Chapter II Responsiveness

A statement of the ability to satisfy the restrictive conditions or minimum standards given in the specifications (see Part I—The "Specifications Module").

Chapter III Equipment configuration and costs

A table (Figure 2) showing all components of

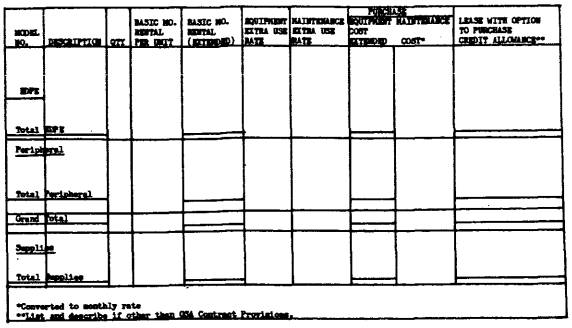


Figure 2. Equipment Configuration and Costs.

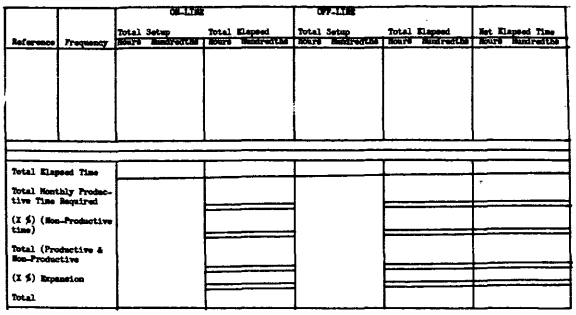


Figure 4. Timing Estimate.

the proposed system with complete pricing information.

Chapter IV Systems and Timing

1. An explanation, with flow charts, of the vendor's approach to the solution of the problem, when the approach constitutes a departure from that given in the specifications. (In some cases, vendors are asked to design or redesign a data system.)

2. Tables (Figures 3 and 4), called "Timing Estimate—Detail" and "Timing Estimate—Summary." These tabulate all time required by the main processor and associated peripheral devices. When these tables are completed properly, it is easy to compare them with the "Workload Statement Chart" (Figure 1) and to determine whether or not each step has been timed. There must be line entries on the "Timing Estimate Charts" for corresponding entries on the "Workload Statement Charts."

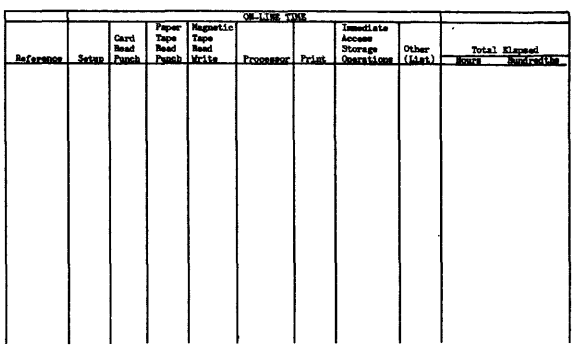


Figure 3. Timing Estimate Stated Workload.

Chapter V Questionnaire

A questionnaire tailored by the requestor to the requirements of the submitted specifications and characteristics of each component structions make it clear that every question must be answered, and that each question will be answered with a number, a percentage, a price or other such specifics.

Chapter VI Equipment Factors

Technical literature giving detailed descriptions and characteristics of each component of the proposed configuration. Requirements for site preparation should be made known here.

Chapter VII Programming and Software

A list and explanation of the software packages *available* for use on the proposed equipment, with emphasis on those particularly useful in the work to be performed.

Chapter VIII Support

1. A description of the training available which meets the requirements of the prospective user.

2. A general discussion of the proposed maintenance plan, including a specific quotation of required preventive maintenance.

3. A statement about back-up equipment available for use in emergencies.

Chapter IX Contract

A copy of the contract (GSA schedule in the case of Federal Government), and any contractual deviations proposed for negotiation which would apply to the proposal submitted.

When both specifications and proposals are prepared in a standard way, vendors know what

their equipment, if selected, would be expected to do, they can prepare their proposals most efficiently and economically, and a sound basis for objective comparison and selection is established.

PART III

The "Validation Module" also serves two purposes. It provides the technicians with standard guidelines for determining the proposal's compliance with the specifications and its accuracy and adequacy. The validators are not charged with determining which of the proposed equipments is best but with determining whether or not the proposals being validated are acceptable. They do not compare one proposal with another.

First the specifications package and the instructions (tailored for the particular acquisition) to the vendors are read. Next the proposals are examined. After the validator is familiar with the specs and the proposal which he is to validate, he is ready to check all statements, answers to the questionnaire, tables and charts in a vendor's proposal. Claims about storage capacities, simultaneity, time estimates, and special purpose devices are some areas requiring detailed consideration.

The validator must have at his disposal a comprehensive library of other publications from the vendor whose proposal is being checked, recognized technical publications from other sources and a vast storehouse of personal knowledge and experience in both the equipment and subject matter fields.

During this process the validator may determine that a proposal does not comply with one or more of the requirements, is not adequate to accomplish the tasks described in the specifications, or that some inaccuracies exist in the proposal. When any such determination is made, the validator is required to prepare a statement containing the specific area or portion of the proposal considered unresponsive, inadequate or otherwise in question. This statement, with recommendations for correction or elimination from further consideration, goes to a reviewing authority for decision. If the reviewing authority concurs with the validator's statements concerning non-compliance or inaccuracies, the submitting vendor may be called in for consultation and mutually agreeable changes.

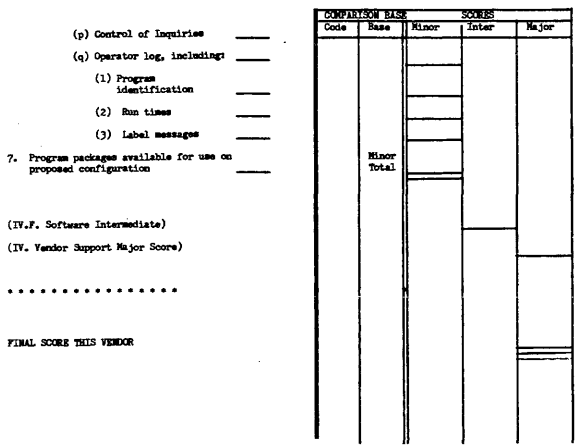


Figure 5. EDPE Evaluation Sheets.

This module also contains the first pass of the "EDPE Evaluation Sheets" (Figure 5). A complete set of evaluation sheets is processed for each proposal submitted. These sheets will already have been modified to reflect the specific list of factors to be considered for the acquisition at hand. The number and types of factors are, for all practical purposes, infinitely variable. There are only two restrictions on the types: (1) Each must represent an item having a bearing on the applicable merits of a proposed configuration; and (2) it must be numerically rateable. There is no allowance in this system for evaluating the color of a sales engineer's tie, or the "reputation" of his employer, or "The one I got was a lemon—they're all no good" line. The tailoring could have been done as early as the time the specifications were sent to the vendors, but may be done at any time prior to receipt of proposals.

The validators, who do no actual scoring, post all entries in the factors section of these sheets. Every blank must be filled in with a number, a check ($\surd$) for yes or no, an N/A or other appropriate specifics as instructed. The basic sources for all data entered, except that resulting from computation, are the proposals and available technical publications.

At the end of this phase the evaluation sheets contain data which is accepted as accurate and complete. These sheets then become the inputs to be compared and scored.

The validators must also prepare a written summary of each proposal. These reports must include actual costs and processing times, and differences between the proposal and the specs with respect to systems approach.

We realize, that for any given selection, those determining which factors to consider may overlook some items. A validator might feel that the list of factors completely ignores some valuable features of a particular proposal. In the report he may discuss any pertinent factors found in that proposal which are not reflected in the evaluation sheets. The report may also be used as a medium for a "sales pitch" for or against a proposal. This report, with the scores, will be considered in the final analysis by those making the selection.

PART IV

The "Scoring Module" is a technique for scoring validated proposals. Factors, already discussed in Part III, relative weights, and detailed instructions for the scoring, all designed for each specific selection, must be completely documented before proposals are in. While each selection to be made may be based on a different set of factors and different weights, the method of applying these weights does not vary.

The inputs to the scoring scheme are the "EDPE Evaluation Sheets" (the factor section completed) and the weights and instructions for their application. The output is an "EDPE Evaluation Summary" (Figure 6).

The scorers accept the validators' entries on the input side as gospel and are only concerned with comparing and scoring, not with validation. The scorers are required to complete the comparison base section of the evaluation sheets and to compute the minor, intermediate, major and total scores as dictated by the weights and formulae supplied. This approach permits a rapid, unbiased selection, and the scorer's data added to that of the validators provides the much needed, formalized backup documentation sure to be required by the final authorities.

CASE NO: _____ MAJOR COMMAND: _____ DATE: _____

PROJECT TITLE: _____

FACTORS	A	B	C	D	E	F	G	H
A. Workload Expandability								
B. Equipment Expandability (I)								
C. Equipment Expandability (A)								
D. Economy of Initial Expansion								
E.								
III. SYSTEM POTENTIAL								
A. Training								
B. Programming/Systems Support								
C. Backup Support								
D. Maintenance								
E. Program Test								
F. Software								
IV. VENDOR SUPPORT								
TOTAL VENDOR SCORE								

EQUIPMENT SELECTED TYPE _____ VENDOR _____
APPROVAL AUTHORITY _____

Figure 6. EDPE Evaluation Summary.

Let's look at a hypothetical set of factors. There might be any number of major categories, including overall cost, equipment characteristics, system potential, vendor's support and others as required. Within the cost category we might look into the actual cost of the equipment on both a lease and a purchase basis, the costs of operating, and installing. The characteristics checked might cover: capacities for program and data storage; compatibilities within the equipments proposed and with data banks; reliability; and total time required to process the whole job spelled out in the workload statements. System potential could cover the investigation of the capability to expand the workload with or without additional equipment. Important aspects of vendor's support might be in the training, maintenance and software areas.

This is a good time to explain what appear to be contradictory statements. In discussing the validation phase we said we do not rate the vendor's reputation. In this phase we say we might rate reliability. Reliability in this sense means the designed methods of insuring no errors or provisions for error detection and correction. There could be three types of reliability insurers—validity checks (does each bit structure represent a "legal" character in that system?), parity checks (does each character contain the proper number of bits?), and accuracy checks (does each character moved

compare with each character to be moved?). A possible accuracy check which no one yet employs is optical scanning of a printed line on high speed printer output and a comparison with the data that was to be printed and is still stored in an output buffer.

We will next develop the "Comparison Base" entries. For each factor to be compared we select the "best" vendor's figure. In the case of costs the lowest is obviously best as is the highest figure in speed areas. The best is then entered on the appropriate line of the evaluation sheets. It is now possible to compare each vendor's figure with the best figure proposed for each item.

Before we go into the actual scoring operation we will discuss the weighting pattern. We will not presume here to even suggest what the weights should be for any factors. The user's requirements alone are the governing element in establishing relative weights as they were in developing the factors.

The maximum score possible for any proposal is 100 points. This score, by the way, can only be achieved if one proposal is "best" in every factor—bar none! We take a deep look at the major categories of factors and assign a part of the 100 to each based on their relative importance. We then determine what part of the maximum score possible for each major category is proper for each intermediate within that major. We then equate the intermediate maximum possible to 100 and assign a portion of this to each minor within each intermediate.

Another way of putting it is: the sum of the maximum points possible for all the major categories equals 100. The sum of the maximum points possible for all the intermediates within a major equals that part of 100 assigned to that major. The sum of the maximum points possible for all the minors within an intermediate equals 100% of the maximum possible for that intermediate.

Now that we understand the weighting pattern we can look at the scoring itself. All of the minor scores for a proposal are developed, then the intermediates, the majors and the total, in that order. Minor scores are developed by either of two methods. Some minor factors are not compared, one proposal with another, but warrant a predetermined score if certain conditions exist. For instance, if free maintenance is provided 24 hours per day, "X" points may be awarded, if only for 8 hours per day, a lesser score is assigned, and so forth.

The second method applies to the majority of the minors. If the SMALLEST is BEST, the score is computed with formula A:

$$\frac{\text{(Comparison Base) (Maximum Possible)}}{\text{("This Vendor" Entry)}} = \text{Minor Score}$$

If the LARGEST is BEST we use formula B:

$$\frac{\text{("This Vendor" Entry) (Maximum Possible)}}{\text{Comparison Base}} = \text{Minor Score}$$

The next step is to total all minor scores within an intermediate in preparation for calculating the intermediate scores. These are computed in this way:

$$\frac{\text{(Minor Total for "This Vendor") (Maximum Possible)}}{100} = \text{Intermediate Score}$$

The intermediate scores within each major category are totaled to produce the major scores and the final scores simply require the summing of all the major scores. The scoring hierarchy, then, looks like Figure 7.

As shown, the final score is derived from the major scores, the majors from the intermediates and the intermediates from the minors.

The final task of the scorers is to transcribe the intermediate, major and final scores from

each evaluation sheet set to the EDPE Evaluation Summary (Figure 6). The completed summary and the validators' reports, discussed in Part III, become the basis for an intelligent, educated decision by a selection group.

Several built-in safeguards further guard against the introduction of personal prejudices and political interests in the entire package from the writing of specifications through the recommendations by a selection group to the final authority. These tricks of the trade may not be obvious in what we have said so far so let us point out a few:

1. Those who write the specifications (the ultimate user of the selected equipment) have nothing to do with scoring the proposals. They are consulted and their suggestions may be incorporated in establishing the factors and relative weights.
2. Those who validate proposals never know the weights assigned to any factors.
3. Those who score do not know which vendor's proposal they are processing. They really don't process a proposal at all. They score a coded set of evaluation sheets.

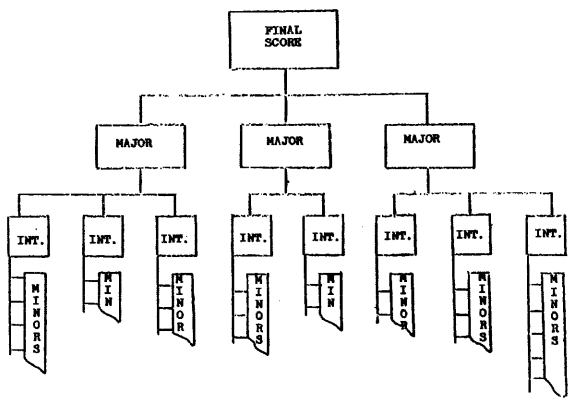


Figure 7. Scoring.

4. Those establishing factors and weights can not be influenced by proposals received since their work must be done before proposals are due.
5. Those responsible for the final decisions have a sound basis for their decisions and might find it difficult to pick anything but the best overall system proposed.

CONCLUSION

The Analytical Technique, as described, is an integrated system which addresses itself to all of the important steps leading to an acceptable selection. Other methods of evaluating individual hardware systems have been designed. These other methods could be used as most valuable aids during the validation of proposals. Such assistance can serve to make the validator's work less tedious and more accurate. At least one system we know mechanizes a good portion of this area and might be especially useful in checking the timing estimates proposed.

Other weighted factor schemes for selection are in use today. Unfortunately, these are not necessarily based on actual requirements, nor are scores necessarily computed the same way for all selections.

Standardized data systems specifications and vendors' proposals, comprehensive validation, and standard computational procedures, are treated in detail and as a single package in our technique.

It is possible to mechanize the scoring portion but the short time required to score manually does not seem to warrant the programming and debugging effort.

This technique has been used a number of times by the Air Force, and it has provided valid selections based on requirements.

COST-VALUE TECHNIQUE FOR EVALUATION OF COMPUTER SYSTEM PROPOSALS

Written by
Edward O. Joslin
EDP Equipment Selection Office (ESQ)
L. G. Hanscom Field
Bedford, Mass.
Edited by
Martin J. Mullin
EDP Equipment Selection Office (ESQ)
L. G. Hanscom Field
Bedford, Massachusetts

INTRODUCTION

The Cost-Value Technique of computer evaluation evolves from the computer evaluation and selection techniques used during the last twenty years. The Cost-Value Technique has benefited by the frustrations, errors, and weaknesses of countless selections; however, each selection has added some knowledge to the growing problems of computer acquisition. Each forward step was accompanied by a technique that was more realistic and meaningful. I consider the Cost-Value Technique a natural consequence of evolution, the successor to the Weighted Factors Selection Method, or any other known existing evaluation technique.

The Cost-Value Technique is an evaluation technique which attempts to make the function of computer selection more meaningful and understandable by eliminating some of the major weaknesses found in most of the evaluation techniques which are in use today. This is not to say that the Cost-Value Technique of computer selection eliminates all the problems found in any other type of objective evaluation and selection technique. All it tries to do is make the difficulties a little easier to overcome, the values assigned a little more realistic, and the resulting selection a little more meaningful and understandable. It does this through the incorporation of two important principles:

1. First, in an effort to keep the selection simple and straightforward, the Cost-Value Technique recognizes only two categories of factors:
   a. Costs; which are a function of the equipment costs multiplied by the time required to complete the application, and many other cost items, and
   b. Extras; which are any items of value that are inherent in the costs of one system proposed, but not to all systems proposed, and do not directly influence the system's running time. Thus, extra maintenance service or extra expansion capabilities which are procured as part of the basic system are extras, but equipment characteristics like speed of central processor and peripheral

equipment, or memory size, are not extras since they directly influence the running time which in turn directly influences the cost of the system.

Thus, an item is only evaluated once, either by its influence on cost—directly or via the running time of the system—which in turn is a cost factor, or by its value as an extra.

2. Second, by using dollar cost or value as the basis for scoring all the 'extras' offered, a common denominator is obtained by which values can be understood, discussed, and changed independent of each other. Also, the 'extras' scores can be directly related to the 'costs' scores.

The significance of these two principles might be better understood if before going into a discussion of the Cost-Value Technique, we took a minute out to consider objective evaluation techniques in general, and the Weighted Factors Selection Method in particular.

Major Premises and Purpose

Objective evaluation techniques for computer selection are based on the following three premises:

1. The 'costs' considered in the evaluation process should be those costs which are associated with securing and maintaining the computer system equipment *and* the support necessary to satisfy the requirements of the specified applications.
2. The major 'value' sought in any computer selection technique is the ability of the selected computer to satisfy the requirements of the applications specified.
3. Some important 'extras' of value such as: excess expansion capability, back-up available across the street, proven 99.5% reliability, etc., will also be included by some vendors in the costs of the system's equipment and support proposed to satisfy the specified application requirements. These 'extras' will be offered in varying amounts by the various participating vendors. These 'extras' are important to the selection and, therefore, their 'value' should be evaluated.

The purpose of any objective evaluation technique might be compared to a series of weight measurements on a set of balance scales. For each 'weighting' (evaluation) the 'costs' necessary to obtain the computer system equipment and support necessary to satisfy the requirements of the specified application, as proposed by each vendor, are weighted against the 'value' of having the application completed *and* the value of the extras offered by that vendor. The objective of the comparative evaluations is to select the 'weighting' (evaluation) that shows the scale tipped most in favor of the 'value' side.

These are the same three premises and purposes that are used in the Cost-Value and the Weighted Factors Selection Method or any other objective evaluation technique. The difference between techniques is in the ways in which these extras are evaluated.

Weighted Factors Selection Method—Pro and Con

The Weighted Factors Selection Method is one of the better evaluation techniques in use today. It recognizes the need for evaluating the 'extras' as well as the standard cost items and does so by assigning numerical values or point scores to all items, 'extras' as well as systems costs. In my opinion, however, there are still two major weaknesses in the Weighted Factors Selection Technique:

1. The technique is likely to give an absolute weight or score to too many factors and then to score the details within each factor in too many different ways, i.e., points for speed of central processor, points for speed of input/output devices, points for time required to complete the full job, points for slack time remaining for expansion, etc. Thus, the same item (i.e. speed) may be awarded points in a number of different ways and for many seemingly different reasons. Soon it becomes very difficult to determine the true worth or influence of that one item on the final selection.
2. The assignment of point scores to specific elements of an evaluation would probably accurately reflect the experiences and knowledge of the evaluator who selected

the elements in either ascending or descending order of importance and then associated each with a fixed numerical value. Arbitrary, perhaps, but what else may the evaluator rely on save what he has learned from experience to be important or, conversely, not important? And, moreover, how can he avoid giving that specific measure of value to elements of the evaluation that reflect in his personal, subjective opinion, their true worth? And, how does he assign scores and identify important elements? An example of such a point assignment is shown in Figure 1. So subjective is this system that "X" number of evaluators, if requested to assign point scores to a list of elements considered important to a computer selection, would produce "X" number of lists; each selective, each the best from the evaluator's point of view, and each different.

COSTS (Total for Five Years)			
EQUIPMENT CHARACTERISTICS			200
Speed		40	
Instructions	10		
Peripheral Equipment	30		
Capacity		40	
Main Memory	20		
Random Access	5		
Magnetic Tape	10		
Peripheral Equipment	5		
Compatibility		20	
Program	10		
Tapes	5		
Cards	5		
Switchability		10	
Magnetic Tape	5		
Printers	3		
Other Peripherals	2		
Reliability		60	
Special Features		10	
Problem Timings		15	
Central Processor Limited	2		
Input/Output Limited	8		
Balanced	5		
Other Characteristics		5	
EXPANSION POTENTIAL			200
Slack Time		150	
Central Processor	50		
Peripheral Equipment	100		
Maximum Expansion		50	
Memory	25		
Peripheral Equipment	25		
SYSTEM SUPPORT			100
Program Assistance		10	
Development	5		
Writing	3		
Converting	2		
Training		10	
Maintenance Offered		10	
Program Testing		20	
Existing Software		50	
Sort/Merge	10		
COBOL	10		
FORTRAN	10		
Report Generator	10		
Other	10		

Figure 1.

Minor differences would be understandable, but the differences are not likely to be minor. The magnitude of the differences can be glimpsed in the startling contrasts of two distinct groups which are presently working their way to management positions. One group has a background in Finance and the other in Engineering. If these two groups were asked, independently, to use the Weighted Scoring Technique to evaluate a computer selection, I suspect the results would be very interesting. Joined as they may be in a common cause, identifiable principally, perhaps, with corporate well-being, they still would regard the method in much different ways and in all likelihood the major weights assigned by these two groups would differ considerably. Shown in Figure 2 is an example of the way these two groups might assign weights to the four major categories listed in Appendix I.

> How do you reconcile such differences of opinion? Moreover, does any reconciliation or compromise improve or enhance the acceptability of resultant 'weighted scores'?

The difficulty—weakness—if you prefer, in the Weighted Factors Selection Method is that points assigned to the cost section cannot be

SAMPLE WEIGHTINGS OF MAJOR CATEGORIES

Categories	Group With Financial Background	Group With Eng. Background
Cost	85%	25%
Equipment Characteristics	5%	25%
Expansion Potential	5%	25%
System Support	5%	25%

Figure 2.

reconciled to the points assigned to the equipment characteristics section or to any other section, or even from item to item within a section. In other words, as the four categories presently exist in the Weighted Factors Selection Method, there is no common denominator between categories, thus there is no meaningful way of relating points from one category to another, or for that matter, between the elements within a category.

COST-VALUE TECHNIQUE—GENERAL

The Cost-Value Technique, like the Weighted Factors Selection Method, also recognizes the necessity of evaluating the 'extras' offered in various computer systems. The Cost-Value Technique, however, is unique in its handling of these 'extras.' The Cost-Value Technique studies any extras offered in the proposals to determine whether the claimed 'extras' are truly important extras or mere incidental elements that appear to be extras. For instance, a sixty-nanosecond memory and a ten-thousand-cards-a-minute reader in themselves are not important extras, if extras at all; the important extra is how much slack time exists in the proposed system because of these high-speed units. For every 'extra' considered important, a study must be initiated to determine the value for the extra. The really distinguishing feature of the Cost-Value Technique is in the assignment of the value associated with these important extras. The value assigned is in terms of costs (dollars).

A brief explanation may be helpful at this point because the significance of the value assignment in cost may not be readily apparent. By 'simply' assigning a cost-value to the various important 'extras' offered by the various vendors, a technique has been found to overcome the seemingly insurmountable problem (no objective method of relating points assigned the various items evaluated) of the Weighted Factors Selection Method. A common denominator has been established with which all offered extras may now be related. Although the cost-values assigned to the various extras will still be a matter of each individual selection and will continue to reflect the opinions of the cost-value assigners, a value when assigned can be understood, examined, discussed, and changed independently of all other individually assigned values. Another very important benefit derived by the use of cost assignment of value in the Cost-Value Technique is that management, for perhaps the first time in the twenty-year history of computers, can understand what is going on in an evaluation and is able to make informed decisions on the value of any disputed extras.

The cost-values established for the various extras found within a proposal are assigned as 'credits' to that proposal (weight added to the value side). This allows each proposal submitted to be evaluated by taking its total 'out-of-pocket' cost and subtracting the cost-value 'credits' awarded that proposal to find the difference. The proposal having the smallest difference (since the 'value' of having the application completed remains constant for all vendors) automatically becomes the proposal selected by the evaluation technique as being most advantageous in terms of the amount of systems equipment and support that is being obtained for the money spent.

An example of how the Cost-Value Technique might be applied is shown in Figure 3.

COSTS		CREDITS	
One-Time Costs		Expansion Potential	
Site Preparation		First Time Block	[illegible]
Electrical	5,000	Second Time Block	[illegible]
Construction	10,000	Third Time Block	[illegible]
Equipment Installation	0		
Equipment Transportation	5,000	System Support	
Vendor Support		Personnel	[illegible]
Personnel	140,000	Training	[illegible]
Training	0	Existing Programs	132,000
Existing Programs	0	Backup Facilities	10,000
Backup Facilities	0	Documentation	15,000
Documentation	0	Program Testing	50,000
Program & Data Conver.	3,000		
Program Testing	0		
		Other Extras	
Continuing Costs		Timeliness	
Computer System Equipment		Inquiry	30,000
Central Processor	915,000	Reports	10,000
Peripheral Equipment		Desirable Compatibility	15,000
Magnetic Tape Units	220,000	Purchase Option	78,000
Card Reader	50,000		
Printer	120,000		
Card Punch	43,000		
Optical Scanner	150,000		
Controllers	332,000		
Auxiliary Equipment	100,000		
Operation and Maintenance	130,000		
Personnel			
Manager	55,000		
Analysts	100,000		
Programmers	400,000		
Operator	35,000		
Others	20,000		
Program Development	0		
Supplies			
Magnetic Tape	12,000		
Printer Paper	10,000		
Cards	7,000		
Total Cost	$2,862,000	Total Credits	$400,000

DIFFERENCE

Total Costs	$2,862,000
Total Credits	400,000
	$2,462,000

Figure 3.

Having now considered the three major premises, the purpose of an objective evaluation technique, and the distinguishing feature (value assigned in cost terms as credits and fewer factor categories used) of the Cost-Value Technique, let us proceed to a detailed examination of the previously mentioned cost-value study of all extras offered.

COST-VALUE STUDY—WITH EXAMPLES

The Cost-Value Technique's approach to the 'extras' offered by the vendors is to appraise the offered extras to determine whether they are worthy of inclusion in the evaluation, and if so, to determine the cost-value of these extras. To avoid any bias, or appearance of bias, on the part of the evaluators, this study should preferably be initiated before the proposals are received. It thus becomes necessary to deal with hypothetical or realistically anticipated extras. A sample listing of items sometimes considered as 'extras' will be helpful to our study and for that purpose a sample listing, similar to one that might be found in the Weighted Factors Selection Method of evaluation as shown in Appendix I will be used. The sample will be used as a beginning for our study of the essentials and extras offered in computer system proposals.

The list shown in Appendix I is arranged into four categories: Costs, Equipment Characteristics, Expandability Potential, and System Support. Other groupings of items could also be used. Some of the items, like those under costs, are not extras, but are included because they must be evaluated in selecting a computer system. In the following sections, each of the listed categories will be studied; one of the results of our study will be the identification of these items and categories that are considered important in the Cost-Value Technique of computer selection.

Before examining each of the four categories, shown in Appendix I, these words of caution and care are furnished.

1. Methods described herein for cost-value determinations are by no means the only methods that could be used.
2. There is nothing sacred in any of the cost-values established in this report since the value of any item depends upon the likelihood of the user's need for that item. For example:
   a. If the described system is to be used only for one or two applications, and the size and volume of these applications are fixed, then the cost-value of Expansion Potential is likely to be nil. On the other hand, if the described system is the first system to be installed in a growing company, the cost-value of Expansion Potential will be very high because every hour of available expansion might be regarded just as valuable as each hour in actual use.
   b. If the described system is to replace an existing, compatible system, the cost-value of some of the System Support items like 'personnel loaned' or 'program assistance' may have no cost-value. If, however, the computer is for a relatively inexperienced group, 'personnel loaned' or 'program assistance' might each have a cost-value as high as, or higher than $20,000 a man year.

Another general word of explanation before getting into the details of cost-value assignment. It will be of help to discuss the idea of cost-valuing the extra. Thus, if four vendors, A, B, C and D, respectively, offer 100 hours, 125 hours, 50 hours and 70 hours of program checkout time, then the amount of 'extra' to which a cost-value should be affixed is 50 hours for A, 75 hours for B, and 20 hours for D. The extra for each item considered for each vendor is the amount offered by that vendor for that item, minus the minimum amount of that item offered by any vendor. If the extras offered are in different terms: Vendor A offers 100 hours of prime shift time, prior to delivery for checkout; Vendor B offers 125 hours, any shift, after delivery, for checkout; Vendor C offers 50 hours of third shift time, prior to or after delivery, for checkout—then all the items have to be converted to their cost-value first, and then their difference taken, to determine their 'extra' cost-value.

The four categories found in Appendix I and to be considered now are Costs, Equipment

Characteristics, Expandability Potential, and System Support.

Cost Items

All cost items should be considered in the evaluation. Items such as the cost of supplies or personnel costs may prove to be non-differentiating (no significant difference found between vendors) in a given selection, but they should not be deleted from the evaluation list until their costs have been found to be truly non-differentiating.

Treating cost items as one-time costs or continuing costs is a matter of cataloging. Two rules must govern any proper treatment of cost items: The costs must be spread proportionately over the expected life of the system, and the system costs must change to reflect the costs of any planned system expansion. If, for example, the life of a system is set at six years and if a uniform expansion rate of 10% a year is expected over the life of the system, then each of the cost items on the list should be charged (if applicable) to the yearly system cost for six years; and it would be expected that the equipment cost for the sixth year would be larger than the equipment cost of the second year.

Another important consideration relating to cost items is that they should show the cost for individual pieces of equipment to be used. This should be done in all cases except when the system is to be used for less than one shift or when the entire system is to be purchased. The break-out shows which equipments are actually to be used for more than one shift (higher rental). The breakout also indicates which items might be more favorably leased than purchased under a split acquisition.

No cost items should be duplicative; that is, the system should not be charged twice for the same equipment or service. For example, if a card reader is used both on-line and off-line, the full cost of the card reader should not be shown twice. Similarly, program development, if performed by users—not contractor personnel—and personnel cost should not both be costed.

Basically, the thought behind the cost items can be summed up by saying, "Any differentiating costs that exist between systems, should be recorded for evaluation."

The second major category of items given points in the Weighted Factors Selection Method was Equipment Characteristics.

Equipment Characteristics

The Cost-Value Technique does not consider any equipment characteristics, in themselves, to be important extras. Instead, their significance is measured in terms of the running time of the system which in turn determines the system's cost and expansion potential.

Typical of the kind of equipment characteristics now being discussed are: the relative speeds and capacities of the systems; hardware compatibility; switchability; reliability; and some other special features. These obviously are features (extras) that determine the time required to complete the applications specified and in turn determine how much slack time is available for expansion, and as such, will be evaluated in the next section on Expansion Potential.

Sample problem timing items (time required to perform some specific set of sample problems) should not be evaluated. They should be used exclusively for the validation of the application timings quoted in the proposals.

Other characteristics (size, weight, etc.) either are included in the space requirements, which are costed; or will appear as special items under a new category, 'Other Extras.' This new category is necessary when using the Cost-Value Technique because many extras to be evaluated do not fit under any of the existing categories.

The third major category given points in the Weighted Factors Selection Method was Expansion Potential.

Expansion Potential

The Cost-Value Technique looks at Expansion Potential in a new way, which is thought to be more meaningful, and evaluates the category by considering the value of the extra offered.

To estimate the Expansion Potential of a system, it is necessary first to calculate the run-

ning time required by the system to complete all required applications. And, at this point, we are compelled to list the elements and aspects of a computer and its use that constitutes its running time. The items that must be considered in any calculation of the running time of a system are listed below:

1. Speed
   a. Central Processor
   b. Peripheral Equipment
   c. Auxiliary Equipment
2. Capacity
   a. Central Processor
   b. Peripheral Equipment
3. Special Features
   a. Parallel Processing
   b. Simultaneous Operations
   c. Other
4. Reliability
   a. Switchability
   b. Error Detection and Correction Features
5. Preparation Time
   a. Set-up/Take-down
   b. Program Insertion
   c. Media Handling
6. Non-productive Time
   a. Reruns
   b. Program Checkout
7. Software Efficiency
   a. Compiled Languages
   b. Assembled Languages

Central processor or peripheral equipment speeds are not new approaches to establishing the comparative merits of competitive equipment systems. What may or may not be new, but which nevertheless must be considered, is the interaction between the central processor and the peripheral equipment. If the time required to do the processing called for in the run is great enough, it might cause delay in the input and output of the data. Thus, what must be determined is the effective speed of the peripheral equipment when operated in conjunction with the central processor. Also, the effect of auxiliary equipment on the system must be ascertained to assure its adequacy to meet the requirements of the input device.

The capacity of the central processor is important because it determines the number of program steps, plus data that can be accommodated at one time. Normally, the larger the capacity the faster the system because fewer load steps are necessary and because sorting and merging functions can process larger volumes of data per pass. The capacity of the peripheral equipment is also important. The printer is a good example of what is meant here. A printer having 160-print positions may permit 'two-up' printing whereas a 120-print position printer may permit only 'one-up' printing, thus doubling its printing time. Again, a printer capable of printing eight copies may be able to do in one printing run what it would take some 'six copy' printers two runs to do. In both of these cases, the capacity of the specific peripheral equipment would materially affect the total running time of the applications.

The special features offered will usually affect the total running time of the system. Obvious examples of this are parallel processing, buffering, and simultaneous operations. The use of these features usually requires some investigation, but their applicability to calculating run time is immediately apparent. Other special features should be examined to determine if their value is to decrease total run time. When such is the case, their influence on run time should be calculated and used.

The use of the above three items, speed, capacity, and special features, are all relatively straightforward in determining total run time. Now, however, we must determine the influence that the systems' reliability exerts on its total run time. How is this influence to be determined? This area of determination is admittedly an area of 'guestimation' rather than definable fact. However, it is felt that a better 'guestimation' can be made here about its influence on total running time than can be made in the absolute on its proper weighted scoring influence on the total selection. The area of concern here is system reliability. This is made up of a number of sub-items: reliability of individual units, number of units available, switchability of these individual units, the availability of error detection and correction

techniques, etc. The intent of considering the above items is to determine the number of hours of downtime per month that could be expected for each of the systems proposed and the frequency of the errors resulting from the systems unreliability. Historical information, if available, should be used for 'ball park' figures. From these figures, educated guesses can be made of the time required to correct the processing effect of these errors and the number of hours of system downtime during which the system is not available. The withdrawal of these two figures from the total time available of the system gives a new systems time available figure.

The preparation time factors must be determined and added to the processing times of each run for each of the systems proposed. The set-up and take-down times are concerned with the time required to prepare the peripheral device for a run and to take-down completed work after a run. Program insertion time is the estimated time required to insert the program into the system's memory. Media handling time is the time required to change any media (magnetic tape reels, new printer paper, etc.) that must be changed while the run is in process.

The non-productive time for an application must be estimated and its time added to the total system time. Non-productive time includes rerun time due to operator or programmer errors and program checkout time. The magnitude of this factor (usually assigned as a percentage of the productive time) is dependent upon the experience of users' personnel and an estimation of the number of new programs to be checked out each month. This factor could change (should decrease as the system gets older) from year to year.

Another big timing factor, upon which little time has been spent to date, is software efficiency. The efficiency of the software is a very important factor since low efficiency may increase the running time by 25% or more. This factor is still in the 'questimation' stage, but with work reliable efficiency factors could probably be determined for all the systems proposed.

Now that all the items necessary for the determination of total running time, except the application, have been discussed, we will see how this running time is used in determination of the expansion potential.

The determination of expansion potential cost-value is made by determining the value of an extra 'time-block' (increment of time) and deducting from this value the cost of obtaining the extra 'time-block.' The cost-value of succeeding time-blocks should be determined until a zero value is obtained.

The use of time-blocks should probably be explained. Time-blocks are used because additional time on a computer system is recognized to have a decreasing value as more of it becomes available. Thus, the first time-block is of more value than the second or any succeeding time-block. This is because the first time-block is obviously much more likely to be used than any succeeding time-block; indeed, this probability of use is one of the ways of assigning a cost-value to a specific time-block. But, why use time-blocks rather than hourly increments or percentage expansion figures? Any of the three could be used. Time-blocks were used because of their convenience. With hourly increments, a great deal of computation is involved. Percentage expansion figures, while likely to be more accurate, involve considerably more work. The extra work required by hourly or percentage approach is not warranted, since the total result cannot be more accurate than value assignments given and the value assignments regardless of the time method used, are still just estimates. Whichever method is used, the important thing is the decreasing value assigned to later time units.

The determination of this cost-value of expansion potential should only be done once. The expansion considered is the expansion potential in the system after its last expansion phase (last year of stated expansion) has been met. At this point, the future expansion potential of the systems become the 'extra' being measured between systems.

Let us take an example with two variations to illustrate the concept explained above. Assume that two systems in the $20,000 a month class are proposed to handle the applications specified. The running times for the two systems, when extended to include all the items

necessary, are found to be 150 hours a month for system A, and 200 hours for system B. Further, system A requires 60 hours a month of maintenance and system B requires only 40 hours a month.

1. If the application is one in which there is little likelihood of having additional expansion, then time-blocks may be made rather long and their per hour value rather small—say 100 hours per time-block and $50 per hour for the first block, $25 per hour for the next time-block, and $10 per hour for each time-block thereafter. Thus, system A would have a value of (assuming 530 hours a month maximum for both systems) $8700 and system B would have a value of $8200. From these value figures would have to be subtracted the cost of the extra shift machine rental, the cost of the personnel required to operate the system, the cost of any extra equipment necessary, and the costs of the extra system maintenance required. If there was any value remaining, it would be called the cost-value and credited once to the system.
2. If the application is likely to grow (or if there is the possibility of sub-leasing extra time to a second party), then time-blocks may be made relatively small and their per hour value rather high—say 20 hour blocks and $150 or more per hour for the first time-block, with each succeeding block decreased by $5.00 per hour. Again, the cost-value for the two systems would be computed by adding up the value and subtracting the cost to obtain these hours. With a large valuation placed on a time-block, the costs of going to higher speed or to larger numbers of peripheral equipment might be justified, if so, the cost of the higher speed units or extra units would also have to be subtracted. The cost-value remaining would be applied one time as a credit.

The last major category given points in the Weighted Factors Selection Method was System Support.

System Support

The Cost-Value Technique considers the value of the extra offered. There are several methods of assigning cost-values to the System Support items listed in Appendix I.

The simplest and perhaps the best method of cost-value assignment is to simply request the other vendors to quote costs to supply a service equal to what is considered 'best' in each case. Thus, if one vendor offers 24 hours on-site maintenance, and the other vendors don't, it might prove very meaningful to ask the other vendors what the extra charge would be (and then credit the largest cost to the vendor already supplying the item, and the largest cost minus each specific vendor's cost for the item would be credited to each of the remaining vendors). The same type thing could be done for program assistance, personnel loaned, training, program testing, documentation, and special software. However, sometimes the costs quoted would be so excessive that it would not make a fair base against which to award value. For instance, if a user was impressed by some special programming routine, he might well ask the various vendors for the cost of supplying such a routine. But, he might receive answers of tens and hundreds of thousands of dollars, where if he himself were to go out and procure such a routine he would not be willing to pay over five thousand dollars. In such a case, the five thousand dollars should become the base. Thus, in cases where the user places a value on a service, lower than a vendor's cost, this value figure becomes the base for determining the item's value. In some cases, however, i.e., maintenance offered, or documentation, the vendor may not be able to give cost figures for supplying service equal to vendors because he just doesn't have the facilities necessary to give equal service. In such a case, the cost-value of such a service must be individually determined and might be considerably higher than the costs charged by any other vendor. In such a case, this higher cost-value figure should become the base.

These system support items might also be handled by making it a mandatory requirement that specific amounts of these items be supplied. (This method is not recommended since a small vendor may not be in a position to meet some of these requirements, and thus he would be

eliminated for items that should properly be treated as 'extras' or at most specific cost items.)

The cost-value of these items might also be ascertained by the user, by taking each item in turn and truly determining its value to him. If the vendor agrees to loan three programmers for three months, what is this really worth? He will get nine man months of work by an experienced programmer which might be the equivalent of fifteen months of programming by his own people (or a new person added), but it may also represent a very important time saving (ready to receive the system sooner). Here are a couple of cost-value items, the cost-value most closely representing the users needs should be chosen. However, the cost-value should never exceed the cost of having the service contracted by someone else, thus, just because a helping hand (personnel loaned) by the vendor will result in the user being ready for the system installation three months earlier and thus result in a $100,000 a month savings for those three months, this doesn't make the value of the personnel loaned $300,000, because, people of the same caliber might have been hired from a software consulting service for $40,000. This $40,000 would then be the cost-value assigned.

Some items like back-up available and debugging facilities are support items on which the vendors cannot be asked to change or improve, therefore, their cost-value has to be evaluated as the items are proposed. An approach to determining the cost-value of back-up would be to determine the probability of experiencing a catastrophic failure, and then determining the cost associated with carrying on the computer activities on the back-up facilities available. The costs times the probability of its happening should give the probable cost for the various systems. Again, cost-value credits can be made out of these costs figures, by simply taking the highest cost minus a specific vendor's costs to determine his credits. Cost-value determination for debugging facilities could be handled in much the same way.

Some items previously covered in other categories in the Weighted Factors Selection Method have not yet been discussed in the Cost-Value Technique. These items are covered under the next category.

Other Extras

There are many other extras that might be offered by a vendor. Items like desirable compatibility or memory lockout can be handled by determining the costs that will be eliminated by the inclusion of such abilities. Thus, the costs that would have to be paid to convert tapes of one sort to another would be saved if the two systems were compatible; this cost therefore becomes the cost-value of such compatibility, or the costs of the time and trouble that could be saved by the inclusion of a memory lockout device becomes its cost-value.

An important extra that will frequently be found in proposals deals with timeliness. A system proposed may claim to be able to allow management to have access to any information within the file in less than one minute, or to have management reports ready by 1:00 p.m. every day, or etc. In these cases, a study must be initiated to determine the cost-value to management of being able to have one minute access, rather than 10 minute access as proposed for other systems; or of having the reports ready by 1:00 p.m. rather than 3:00 p.m. as proposed for the other systems, etc. Timeliness cost-values will usually have to be established by management (the group to whom the 'time is money' statement has the most meaning), but management should also be required to substantiate their cost-value assignment by showing the saving or advantages that will result.

Another type of extra, and an extra that is perhaps worth all the trouble necessary in making the system study, is the possibility of a new innovation or systems approach. The cost-value to be assigned would be equal to a realistic determination of the saving that would be accrued by using the suggested approach.

Summary of Proposed Cost-Value Technique

We have been briefly exposed to some techniques for determining the cost-value of a number of items that should be included in any selection. The cost-values derived for the various vendors are applied as credits to offset the costs of the system and services he proposed. The vendor having the smallest difference (out-

of-pocket cost minus credits), is the vendor to whom the contract should be awarded.

The Cost-Value Technique of computer selection does not do away with all the problems found in any other type of objective evaluation and selection technique. All it tries to do is make the difficulties a little easier to overcome, the values assigned a little more realistic, and the resulting selection a little more meaningful and understandable. It does this through two important principles:

1. First, in an effort to keep the selection simple and straightforward, the Cost-Value Technique recognizes only two categories of factors:
   a. Costs; which are a function of the equipment costs multiplied by the time required to complete the application, and many other cost items, and
   b. Extras; which are any items of value that are inherent in the costs of one system proposed, but not to all systems proposed, and do not directly influence the system's running time. Thus, extra maintenance service or extra expansion capabilities which are procured as part of the basic system are extras, but equipment characteristics like speed of central processor and peripheral equipment, or memory size, are not extras since they directly influence the running time which in turn directly influences the cost of the system.

   Thus, an item is only evaluated once, either by its influence on cost—directly or via the running time of system—which in turn is a cost factor, or by its value as an extra.
2. Second, by using dollar cost or value as the basis for scoring all the 'extras' offered, a common denominator is obtained by which values can be understood, discussed, and changed independently of each other. Also, the 'extras' scores can be directly related to the 'costs' scores.

I feel the Cost-Value Technique is a big improvement over existing objective evaluation techniques, but it too can evolve. The next section briefly mentions some of the further improvements that might be made in Cost-Value Techniques.

POSSIBLE ADDITIONAL IMPROVEMENTS

A number of improvements might be made to the Cost-Value Technique. The use of debits, as well as of credits, could be used immediately and might make the technique a little more natural. But, two other improvements, time dependent cost-value assignments and quality determination, require more work with and a more thorough understanding of the Cost-Value Technique before their importance can be fully measured.

Use of Debits

The Cost-Value Technique could be improved by the use of debits in addition to credits. With 'debits,' some items such as penalties for failure to meet certain requirements (e.g., late submission of proposals, late delivery of equipment, etc.), could be more easily handled. Also, in the present technique certain desirable items, if present, are awarded credits; however, it might be more meaningful if their absence from a proposal and/or system was at the risk of debits. Thus, credits would be used to reward systems really proposing some valuable 'extras,' while proposals lacking certain features considered valuable in the original system specifications would be negatively recognized by the assignment of debits.

Time Dependent Cost-Value Assignment

A matter of concern in the present Cost-Value Technique is the fact that the costs to be incurred in the future (rental of equipment four years hence) have as much weight and value as the same categories of costs which are to be incurred in the present or the immediate future. Thus, I may be relatively certain that I will be paying $12,000 a month in rental this year, but I am not too certain that I will be paying $12,000 a month five years from now because the system requirements may have changed or I may have a much larger or a much smaller system at that time. Perhaps what is needed to equate such consideration is a time dependent, cost-value assignment. This assignment might be produced by multiplying the

cost of the item by the probability that the same cost will ever be paid. This approach would tend to make one-time costs more significant in the selection and minimize the amount of influence which 'expansion potential' tends to exert on the selection.

Quality Determination

The Cost-Value Technique, as discussed, does not provide a value assignment for the quality of supplied documentation or of the instruction to be given, etc. These quality items are of importance and eventually must be incorporated into the technique. However, much data will have to be collected on such items as these to build a body of experience, to create the limits of their importance, and to relate their importance with other elements of the system.

There are undoubtedly other modifications that could be made to the Cost-Value Technique. But, changes are expected and needed if a technique is to grow. However, whatever changes might be made, they should not affect the basis for the Cost-Value Technique, which is the use of costs as the common denominator when relating the various items that must be considered when evaluating computer proposals. This principle of using costs as the common denominator when valuating items to be included in an evaluation technique is the heart of the Cost-Value Technique, and it is likely to be a basic principle that will be incorporated into most of the new objective evaluation techniques that will evolve over the next 20 or so years.

SUMMARY

The Cost-Value Technique proposes two major changes to existing evaluation techniques.

The first change is in methodology. The Cost-Value Technique attempts to consider all items of value to a computer system, but to consider them only once and in the environment in which they belong. The categories scored are total system cost and 'extras' which are defined as features like expansion potential, vendor support, or similar characteristics which are part of total system cost, but differentiating between vendors.

The second change is in scoring technique. The Cost-Value Technique uses dollars rather than weighted points as the basis of comparison. This provides a more natural basis for comparison. It eliminates the need for 'trade-offs' and gives management deeper understanding of the total selection process.

These two changes are intended to bring about a more understandable and realistic selection.

The Cost-Value Technique is intended to be a dynamic technique to which additional modifications might be made. Some possible future improvements that might be incorporated into the Cost-Value Technique have also been shown.

APPENDIX I

Items Considered in Sample Weighted Factors Evaluation Method

COSTS

One-Time Costs

- Site Preparation
  - Electrical
    - Air Conditioning (Cooling, Heating, and Humidity Control)
    - Power Supply (including all wiring)
  - Construction
    - Facilities (Space, walls, ceiling, painting, draperies)
    - False Flooring (including bracings)
- Equipment Installation
- Equipment Transportation (including insurance cost)
- Vendor Support
  - Personnel
    - Analysts
    - Programmers
    - Operators
    - Instructors
  - Training (including cost of transportation and living costs, if training not provided on-site)
  - Existing Programs
  - Back-up Facilities
  - Machine Time (checkout)
  - Documentation
  - Program and Data Conversion

One-Time or Continuing Costs (Dependent upon procurement method used)
- Central Processor and Associated Equipment
  - Central Processor
  - Console
  - Floating Point Option
  - Multiply Option
  - Real Time Option
  - Memory Units
  - Etc.
- Peripheral Computer Equipment—On-Line or Off-Line
  - Remote Inquiry Device
  - Card Reader
  - Card Punch
  - Printer
  - Magnetic Tape Units
  - Immediate Access Storage (IAS) Units
  - Paper Tape Reader
  - Paper Tape Punch
  - Controllers and Buffers
  - Micr, Optical Scanner, etc.
- Auxiliary Equipment
  - Key Punch Machines and other data Creation Devices (flexiwriter, teletype machine, etc.)

Continuing Costs
- Operation and Maintenance of all Electrical Equipment (Above Computer System Equipment, plus Air Conditioning, etc.)
- Personnel
  - Managers
  - Analysts
  - Programmers
  - Operators
  - Others (key punch operators, etc.)
- Program Development
- Supplies
  - Magnetic Tape
  - Cartridges for IAS
  - Printer Paper
  - Cards
  - Programming Forms
  - Etc.
- Indirect Cost-Space Used

EQUIPMENT CHARACTERISTICS

Speed
- Time Required to Complete Applications Specified
- Instructions
  - Add time (fixed and floating)
  - Multiply time (fixed and floating)
  - Divide time (fixed and floating)
  - Move
  - Etc. (through all other instructions though significant)
- Peripheral Equipment
  - Printer (lines per minute)
  - Card Reader (card per minute)
  - Card Punch (card per minute)
  - Magnetic Tape Units (characters per second)
  - IAS (characters per second average)
  - Etc. (through all other peripheral equipment listed)

Capacity
- Characters of Storage in Main Memory (core)
- Characters of Storage in IAS
- Characters of Storage on Magnetic Tape
  - Length of printed line, in characters
  - Card size
  - Length of Paper Tape
  - Etc.

Compatibility
- Program
- Tapes
- Cards

Switchability
- Magnetic Tape Units
- Printers
- Other

Reliability
- Error Detection
  - Parity Checks
  - Validation Checks
  - Accuracy Checks
- Error Correction Techniques
- Near-Time-to-Failure (etc.)

Special Features
- Memory Lock-out
- Parallel Processing

Problem Timings—Sample Problems
- Central Processor Limited
- Input/Output Limited
- Balanced

Other Characteristics

Size of Equipment (each piece considered)
Weight of Equipment (each piece considered)

EXPANSION POTENTIAL

Slack Time (Amount of available free time on each piece of system equipment)

Central Processor
Magnetic Tapes
IAS
Card Punch
Printer
Etc. (Through all other system equipment offered)

Maximum Expansion (Number of units that can be added to system)

Magnetic Tapes
IAS
Card Punch
Printer
Etc. (Through all other system equipment offered)

SYSTEM SUPPORT

Program Assistance

Development
Writing
Converting

Training

Analysts
Programmers
Operators

Maintenance Offered

Backup Availability

Program Testing

Hours
Location

Existing Software

Sort
Merge
COBOL
FORTRAN
Report Generator
Etc.

Documentation

Personnel Loaned

Analysts
Programmers
Operators

BIBLIOGRAPHY

Thesis

BIBLIOGRAPHY

1. JOSLIN, EDWARD O., *Computer Acquisition*, Prepared as MBA Requirement for Boston College.
2. GREGORY, R. H. and VAN HORN, R. L., *Automatic Data Processing Systems*, Editions 1 and 2, Wadsworth.
3. GREGORY, R. H. and VAN HORN, R. L., *Business Data Processing and Programming*, Wadsworth.
4. CONWAY, GIBBONS and WATTS, *Business Experience with Electronic Computers*, Price Waterhouse & Co.
5. D.P.M.A., *Data Processing—Volumes IV, V and VI.*
6. WILLIAMS, PERROTT, WEITZMAN, MURRAY and SHOBER, "A Methodology for Computer Selection Studies," *Computers and Automation*, May 1963.
7. GOSDEN, J. A., "The Computer Chooser's Quandary; Which Machine and Why?," *Datamation*, December 1962.
8. CASTILLO-FERNANDEZ, JOSE A., ENRIQUE RIVERA-SANTANA, "Technique for Evaluating Electronic Computers", *Data Processing*, September 1962.
9. SISSON, R. L., "How to be a Comparison Shopper for Computers", *Business Management Magazine*, October 1962.
10. "Better Computer Comparisons for You", *EDP Analyzer*, June 1963.
11. "New Ways for EDP System Studies", *EDP Analyzer*, September 1963.
12. CANNING, RICHARD G., *Selection Procedure, EDP Systems Analysis Technique*, 1962, from Volume I, *Standard EDP Reports* by Auerbach/BNA.

13. FRIEDLAND, E. I., *How to Select the Best Computer: A Conceptual Outline,* July 1963, MITRE Working Paper, W6231.

14. BAGLEY, P. R., *Data Collection for Evaluation of EDP Proposals,* July 1963, MITRE Working Paper, W6283.

15. Rosenthal Committee, *Analytical Technique for Automatic Data Processing Equipment Acquisition,* 1963.

16. *Management of Data Processing Equipment,* "Selection of Data Processing Equipment," Air Force Manual 171-9, March 1962.

17. *Directorate of Data Automation,* "Selection of Electronic Data Processing Equipment," AFADA, October 1, 1962.

THE USE OF A COMPUTER TO EVALUATE COMPUTERS

Donald J. Herman, President, and Fred C. Ihrer, Vice President and Technical Director
COMRESS, Incorporated
Washington, D. C.

THE PROBLEM OF COMPUTER EVALUATION

The complex problem of evaluating and selecting the optimum systems approach for the optimum computer to solve a particular data processing problem, has plagued management since the time that computers came into use for business and scientific data processing.

The review, evaluation, analysis and selection of data processing hardware/software configurations establishes the necessity for relating the functions of hardware performance characteristics and software program efficiency factors to the specifications of the proposed computer application. Each of these functions comprises interacting and interdependent activities throughout the complete evaluation cycle, beginning with the delineation of a data processing problem and ending with a successfully installed and efficiently operating computer. To accomplish this effort through a manual method of making comparisons and estimates of the relative proficiency of the various computers is a time-consuming and costly effort which does not always produce the ultimate solution. The requirement to establish proper relationships for evaluation purposes between each of these functions identifies the need for a management vehicle which will assist management in making a proper and correct computer evaluation decision.

Various attempts have been made at the proper solution of this problem but none of these approaches has incorporated the maximum utilization of the capabilities of a computer. A detailed analysis of the evaluation problem revealed that a series of techniques, incorporating a wide range of scientific disciplines, could be utilized to approach an optimum solution. However, these techniques demanded a prodigious amount of calculation. To attempt to apply them to the problem without the aid of a computer would have been similar to solving complex mathematical problems with the use of Roman numerals.

A COMPUTERIZED SOLUTION TO THE PROBLEM

COMRESS has prepared a software package called SCERT, meaning Systems and Computers Evaluation and Review Technique. SCERT is a simulation program which has been designed to accept the definitions of a data processing problem, and to build a mathematical model of each program run in the defined problem. Also, SCERT maintains a library of hardware and software performance factors for a wide range of digital computers. Using the algorithms which have been incorporated into the program, it can extract the appropriate hardware and software factors for all the components in any one configuration. With this information, it will build a mathematical model representing the hardware/software performance capabilities of each selected computer configuration. During the simulation phase then, SCERT simulates the response of

each of the "program models" against the "performance model" of each of the selected configurations.

The results of this simulation are represented in the form of several different management reports which furnish the user with projections of cost, time, memory and manpower requirements which would be necessary to put his data processing system "on the air" for any one of the computers evaluated.

SOME OF THE USES OF THE SCERT PROGRAM

SCERT has been used during the past fifteen months by a large number of users representing a wide and diversified range of data processing problems and systems. These have varied from the typical business type sequential processing problem to the scientific problem and the real-time random access and communication problems.

Hardware Selection

For the data processing installation making its initial computer selection, SCERT provides management with an extremely valuable tool for assuring the selection of the proper computer hardware. SCERT can be used to evaluate the performance of an infinite number of computer configurations, thus assuring management that they have selected the computer which most economically meets their processing requirements.

Additionally, the SCERT projections will provide the new installation with many valuable guidelines for the implementation of the system on the selected computer. SCERT will realistically project the programming man months involved and data media requirements and will establish accurate running time goals for the completed programs. Additionally, SCERT will aid in systems design by reflecting optimized tape block size, channel assignments, internal running times and memory requirements by function of the computer.

Hardware Enhancement or Replacement

The use of SCERT by an installation which is facing the problem of enhancing or replacing its present computer hardware can provide management with the optimum approach to this recurring problem. SCERT can be used to simulate a variety of potential hardware enhancements such as faster tape stations, additional memory, etc., and will accurately project the differences in program running time for each of the enhancements considered. Additionally, SCERT can be used for considering the impact of replacing present hardware with larger or more modern hardware and will make realistic projections of utilization and cost of reprogramming for the various options considered. It can also be used to determine the effect of running existing programs on compatible machines in compatibility mode situations.

Application Analysis

SCERT has been used by computer installations to evaluate the impact on both their utilization and programming resources when considering the addition of new applications. In this environment it can be used to project the running time of the new application, the utilization of off-line and auxiliary equipment and the programming effort required. It is very frequently used to evaluate several different systems design approaches for the same problem in terms of computer running time and programming effort.

Installation Review

SCERT has also been used by computer installation managers who are interested in evaluating the performance of their installation in terms of the standard projections made by SCERT. Since the SCERT projections are always based on an optimum use of computer hardware available, then these projections provide highly realistic goals and standards by which actual program performance can be measured. This allows management to isolate those programs which are most subject to enhancement or reprogramming.

Hardware Design

When used in the performance of this function, SCERT provides the user with a valuable tool for assuring the capabilities of determining the correct design specifications of computer components, far in advance of committing research and development funds and valuable engineering time. By utilizing this program

during the early planning stages for new hardware, it is possible to evaluate the performance of a wide range of product specification variations to determine the optimum computer characteristics required to meet the demands of a competitive market.

Generally, the procedure used for accomplishing this function is one of defining a series of pre-selected and well defined systems applications to the SCERT program. By then varying the planned performance specifications of a "paper machine," the user is able to determine the performance of various computer configurations, and select the one which best meets the desired market performance requirements.

OPERATIONS OF THE SCERT PROGRAM

SCERT is an extremely complex and fairly large program. The size of the program approaches the magnitude of 31,000 instructions. It contains approximately 5,000 algorithms and maintains a library of approximately 100,000 hardware and software factors.

The program was originally written in a COMRESS specially designed language for an RCA 301 computer. As an operating software package on the RCA 301, it consists of 26 phases and as such, the program would require from two to four hours of running time to simulate a problem consisting of 100 computer runs for the evaluation of six different computer configurations. The running time is a function of the size of the configurations being simulated, and the complexity of the defined programs.

Our experience in defining runs to SCERT can best be measured as a range of magnitude, rather than an average or typical time. This range has varied from a minimum of ten to a high of fifty runs and their associated files defined in one man day of effort. A typical average elapsed time-frame for the complete production of a SCERT analysis is approximately four weeks. The time required for definition is usually a function of the condition of documentation and/or the availability of the time of the systems analysts who designed the system to be simulated.

COMPUTERS MAINTAINED IN SCERT LIBRARY

At the beginning of 1964, the SCERT program included hardware and software factors on each of the following computer central processors and related peripheral devices. New computers are added to the library within one month of manufacturer's announcements.

Manufacturer	*Computer*
Burroughs	205
	200 series
	5000
Control Data Corporation	160A
	1604
	3200
	3600
	6600
General Electric	215
	225
	235
	425
	435
Honeywell	200
	400
	1400
	800
	1800
IBM	1440
	1401
	1460
	1410
	7010
	7040 and 7044
	7070, 7072 and 7074
	7080
	7090 and 7094
	705 II and III
NCR	304
	315
Philco	2000
RCA	301
	501
	3301
UNIVAC	SS 80 I and II
	1050
	U III
	1107
	490

DESCRIPTION OF SCERT PROGRAM

The SCERT program can best be described as the four phase program which is depicted on Figure 1. Phase I builds a mathematical model of the system and assures that it is a systems-oriented model. Phase II adjusts this model to the type of machine which is to be evaluated. Phase III simulates the performance of these models on each of the configurations being evaluated. Phase IV produces the necessary decision-making data on the several output reports of SCERT.

PHASE I

In this phase, SCERT creates a mathematical model of each computer run in the entire system application. The bases on which the model is built is the Systems Definitions, Environment Definitions and the File Definitions.

The *Systems Definitions* (Figures 2 and 3) are basically a verbal portrayal of the systems chart reflecting the frequency and occurrence, the number and identity of I/O files, and the type of internal computer activity.

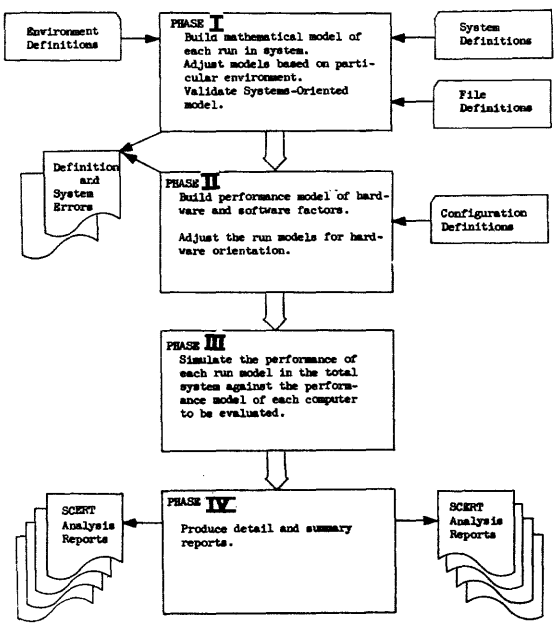


Figure 1. Schematic of SCERT Program.

Several classes of mnemonic codes are used to define internal computer activity; some of these are:

General Job Codes

Sort
Merge
Sequence Memory

Data Processing Job Codes

Update
Extract
Validate

Mathematical Job Codes

Square Root
Sine
Cosine
Arc Tangent
Etc.

Matrix Job Codes

Matrix Multiply
Matrix Transpose

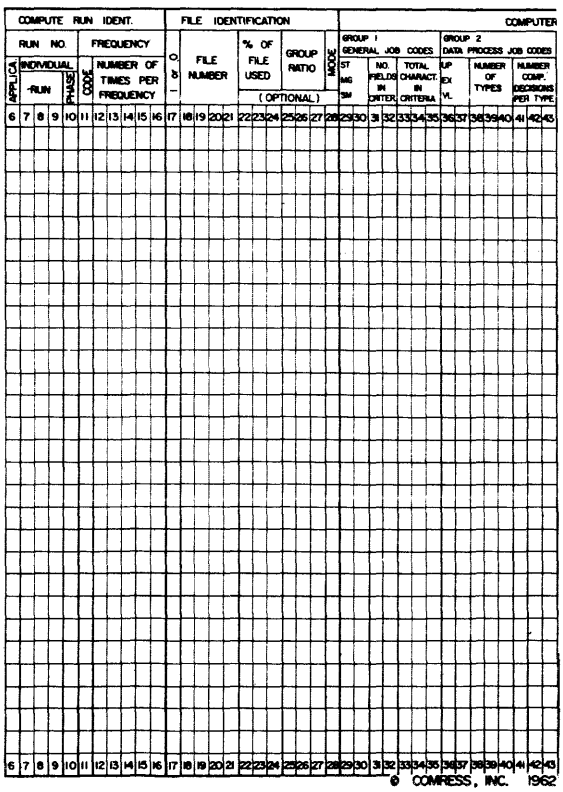


Figure 2. Systems Definitions—Part 1.

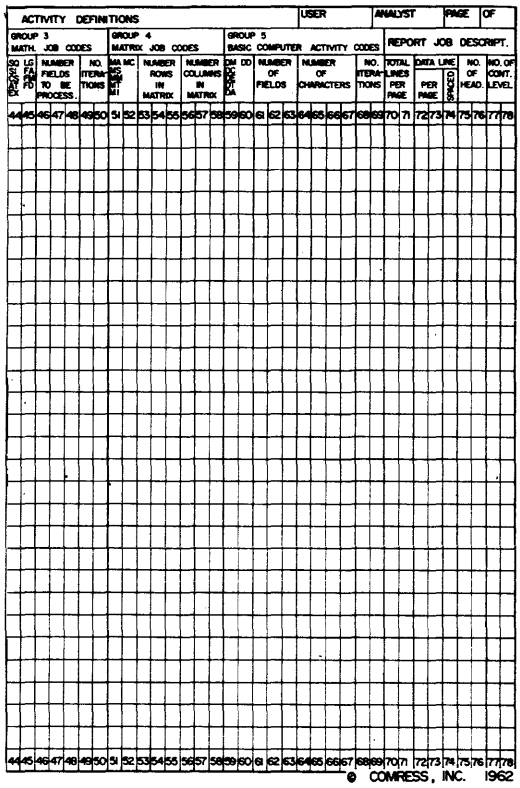


Figure 3. Systems Definitions—Part 2.

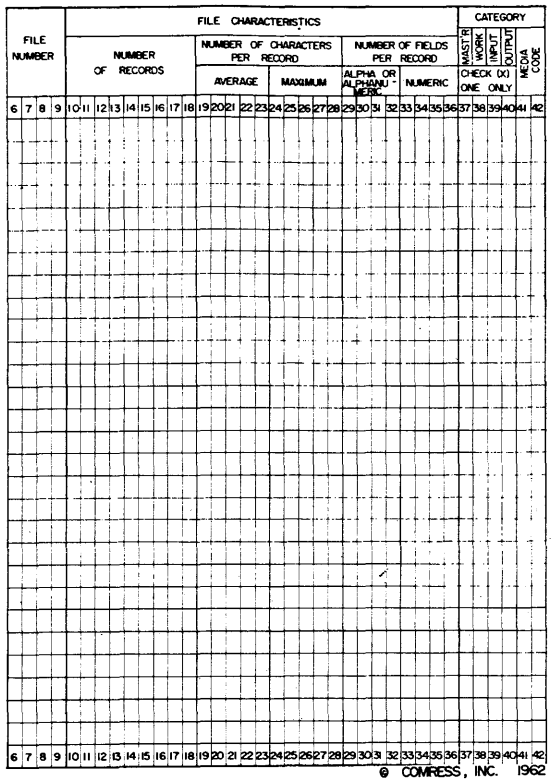


Figure 4. File Definitions.

Matrix Invert
Matrix Subtract
Matrix Divide
Matrix Add

Basic Computer Activity Codes

Data Move
Data Compare
Data Edit
Data Translate
Data Add
Data Divide
Data Multiply
Data Subtract

The *File Definitions* (Figure 4) reflect the basic parameters of each file. These specifications include such data as the number of records, number of characters per record, the number of alpha and numeric fields, and the category and data media of the file.

The *Environmental Definitions* (Figure 5) provide SCERT with the necessary factors or factor adjustments for all functions based on pure judgment. This input phase of SCERT is an open-ended design to allow for the individual requirements of each user.

The types of data which are introduced by these definitions are:

Software Environment
Personnel Environment
General Environment

The Software Environment division provides the user with a method of specifying to SCERT the programming language, the Input/Output Control System, and the Program Monitor System he will use.

The Personnel Environment division defines for SCERT the experience level of the user's programming staff. SCERT mathematically relates this to the complexity of the systems application and the programming difficulty of the language to be used for the configuration to be evaluated.

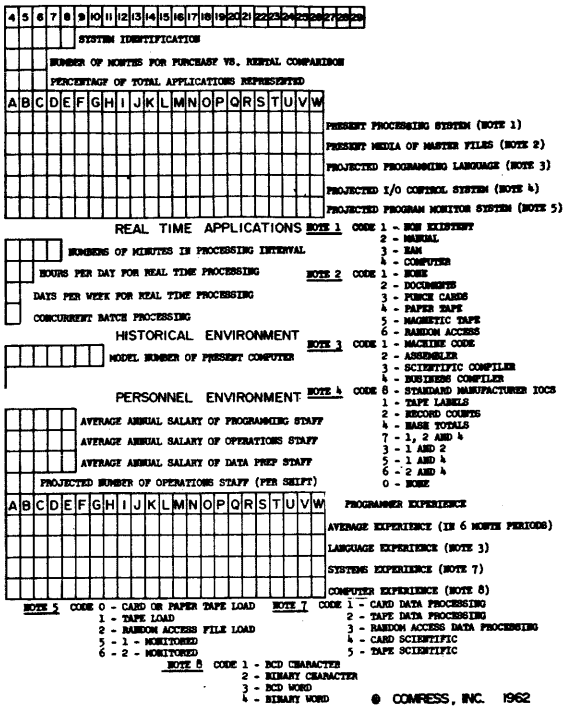


Figure 5. Environmental Definitions.

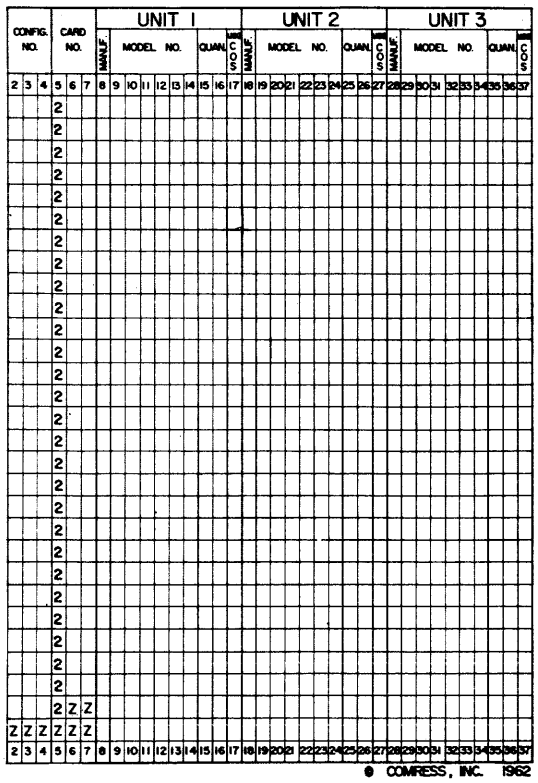


Figure 6. Computer Complement Definitions.

The General Environment division allows the user to specify the length of time he plans to keep the computer, and such other factors as the percentage of the total application that the evaluated system definitions represent.

PHASE II

Phase II of the SCERT program accepts the Computer Complement Definitions (Figure 6) which define each of the various computer configurations to be evaluated. Based on these definitions, it extracts from its library of hardware and software factors, those required for each computer configuration. SCERT arrays these factors in the form of a three-dimensional matrix and then modifies the mathematical models of the runs in the system to conform to the hardware requirements of the particular computer. For example, the record sizes involved are, if necessary, made multiples of the word structure of the computer; and assignment of the input and output files is made to the appropriate peripheral devices available.

The Computer Complement Definitions which are input to this phase are simply an identification of the manufacturers' model numbers, the quantity of each in the configuration, and the function code which specifies how the particular model will be used. The function code can specify that a unit will be operated on-line, as a satellite or in parallel. The definitions must also include the model number of the satellite computer, if any, as well as the model numbers of all peripheral and control devices associated with the satellite.

During both Phases I and II, SCERT performs a complete validation of all input definitions. It checks for proper coding and for consistency of coding. The systems application is continually checked to insure its validity, especially in terms of the computer configuration being evaluated. For example, SCERT will determine if sufficient peripheral devices are available and are of the right kind to handle all

input and output functions. Errors discovered by SCERT in performing these validations are printed and the user has the option of continuing or restarting the program.

PHASE III

The basis for most of the decision theory techniques incorporated into SCERT is the simulation of the performance of each computer configuration for each computer run model to be evaluated. This simulation technique can best be described as the explosion of the computer run into its maximum number of thru-put iterations. SCERT contains over 5,000 algorithms which it uses to perform this simulation. Initially, all input, output and internal computation times are calculated. As these timing forecasts are developed, a preliminary assignment of these times is made, based on the powers and features of the central processing unit. Then, based on the simulation of the thru-put iterations, the simultaneous and net times are derived for each run. As a by-product of this simulation, SCERT computes the number of program steps, amount of memory used, number of tapes required for input and output files, etc. In addition, a pre and post run history is developed to forecast the utilization of peripheral equipment such as key punch, off-line card reading, satellite computer time, and other requirements.

The various terms in the algorithms are products of several sources. The system input definitions furnish a primary source of terms. Another set is provided by the environmental definitions. A third set of terms is derived by SCERT from historical and statistical tables, and another important source is furnished by the program's library of hardware and software factors. Currently, this library contains fifty computer systems which are reflected by over 100,000 different hardware and software factors.

PHASE IV

This phase of SCERT provides the user with various levels of data developed by SCERT during the course of simulation and evaluation computations. The several output reports which are generated by SCERT during this phase have been designed to provide the user with a comprehensive range of detailed and summary management information. These reports represent a digest of several million computations and decisions made in a typical evaluation. The data represented on some of these reports is, to our knowledge, the first attempt ever to have been made to provide the managers of a data processing installation with data which can be used for management purposes. One of the examples of this type of data is shown on the SCERT Detail Systems Analysis report which provides the programmer with much of the information which he would, under ordinary circumstances, have to develop through trial and error methods prior to the writing of his program. For instance, SCERT will optimize for each file in every program the record size, the block size, the input/output media assignment, and the I/O channel assignment.

Computer Complement Report (Figure 7)

This report is provided for each computer complement evaluated and serves mainly as an identification of the computer configuration. It reflects, by type of hardware device, the quantity of each model number in the configuration, the purchase price, minimum monthly rental, and the environmental requirements of flooring, cooling, and the power requirements.

Central Processor Utilization Report (Figure 8)

This report summarizes, by program run, the SCERT forecast for program running, set-up/

SCERT ANALYSIS PART 1 COMPUTER COMPLEMENT REPORT

FOR SYSTEM PARAC, COMPUTER COMPLEMENT NUMBER 21P

MODEL NUMBER	MANUFACTURER	QUANTITY IN SYSTEM	PURCHASE PRICE $	MINIMUM MONTHLY RENTAL	ENVIRONMENTAL REQUIREMENTS FLOORING SQ FT	COOLING BTU/HR	POWER SUPPLY KW	KVA
CENTRAL PROCESSING UNIT								
COMPUTER MAIN-FRAME								
7104A	IBM	01	923100	20045	100	43800	12.0	15.0
MAGNETIC TAPE STATION								
7204	IBM	08	336000	7200	144	31200	18.2	19.8
7211	IBM	02	72000	1488	40	7800	02.5	0312
PUNCH CARD EQUIPMENT								
75011	IBM	01	3100	75	04	800	00.2	00.3
MONITOR TYPEWRITER								
7350	IBM	01	13050	388	10	800	00.0	00.0
THE COST AND OTHER FACTORS FOR THE FOLLOWING FEATURES AND CONTROL UNITS ARE ADDED TO DEVICES ABOVE.								
2295	IBM							
7014	IBM							
7607	IBM							
7026	IBM							
7041	IBM							
7021	IBM							
THE FOLLOWING SATELLITE COMPUTER SYSTEM IS ALSO CONSIDERED FOR UTILIZATION AND COMPATABILITY.								
1401C3	IBM							
1402	IBM							
1402	IBM							
14032	IBM							
7292	IBM							
TOTAL EDP SYSTEM			1341258	29688	332	84488	25.6	32.1

Figure 7. Computer Complement Report.

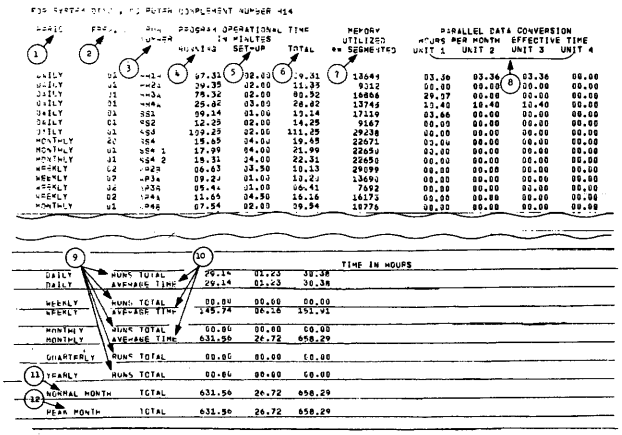


Figure 8. Central Processor Utilization Report.

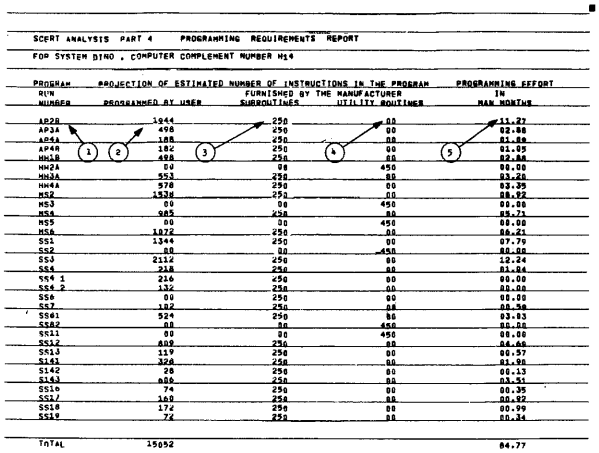


Figure 10. Programming Requirements Report.

take-down time, and memory utilization. It further provides information for those applicable computer systems on the parallel data conversion that could be scheduled during each program run. After listing these forecasts, SCERT then summarizes the projections for daily, weekly and monthly average utilization time, and finally, prints the projections for peak period utilization.

Satellite Computer and Auxiliary Equipment Utilization Report (Figure 9)

Based on the pre and post run history developed during the SCERT simulation, this report forecasts for each computer run in the system, the satellite computer and auxiliary equipment utilization. Auxiliary equipment utilization is subdivided by function such as card to tape, tape to print, and other off-line data conversion operations. Also provided is the number of hours of data preparation work, such as key punch time, required to support each run.

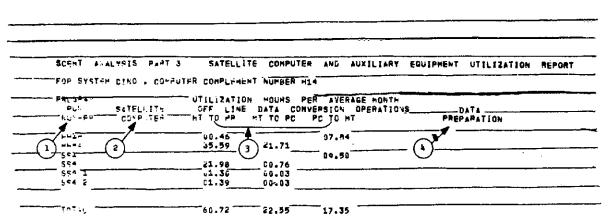


Figure 9. Satellite Computer and Auxiliary Equipment Utilization Report.

Programming Requirements Report (Figure 10)

This report, which is also a recap by individual program run, projects the number of program steps required to put the program "on the air," and then makes an estimate of the user's programming effort in man months. It further reflects the number of program steps saved by using applicable sub-routines and utility programs furnished by the manufacturers. The programming effort in man months projection is a function of the number of steps in the program related to the programming language specified by the user in the Environment Definitions. This computation is made in relation to the experience of the user's programming staff correlated to the programming difficulty of the computer involved, and the nature of the individual program run.

Application Summary Report (Figure 11)

This report is a summation of all program runs within each application area and is provided to assist the user in determining the priority to be assigned to the various applications for implementation purposes. It shows the monthly system utilization by application together with a projection of programming effort and associated cost.

Cost Summary Report (Figure 12)

This report is provided for each computer configuration evaluated. It is a management digest of the costs associated with installing a

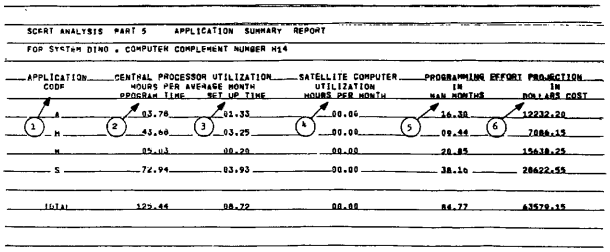


Figure 11. Application Summary Report.

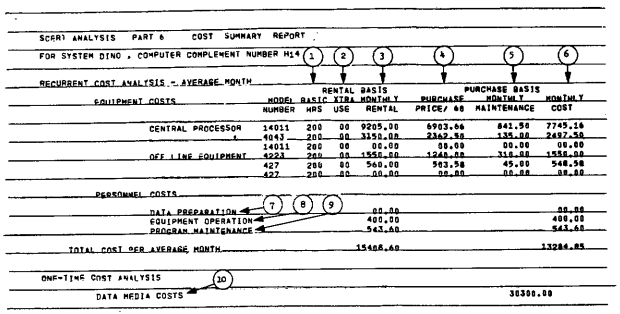


Figure 12. Cost Summary Report.

computer and operating it once all applications are programmed. Recurring costs show the monthly rental computed by SCERT, based on the projections of utilization. If manufacturers have several rental options, SCERT will have selected the optimum. Shown for comparison purposes are the purchase costs which include the cost of maintenance. In addition to the equipment costs, the recurrent cost analysis also reflects personnel salaries for equipment operation and data preparation and program maintenance.

The One-Time Cost Analysis shows the costs of acquiring data media such as magnetic tapes.

Detailed Systems Analysis (Figure 13)

This is an optional three-part report which provides a comprehensive analysis of each computer run as simulated by SCERT. Part I reflects for each input and output file the total buffered and unbuffered time, the input/output device and channel assignment selected by SCERT, the number of reels, if magnetic tape file, and the optimum records per block developed by SCERT. Part II of the report is an

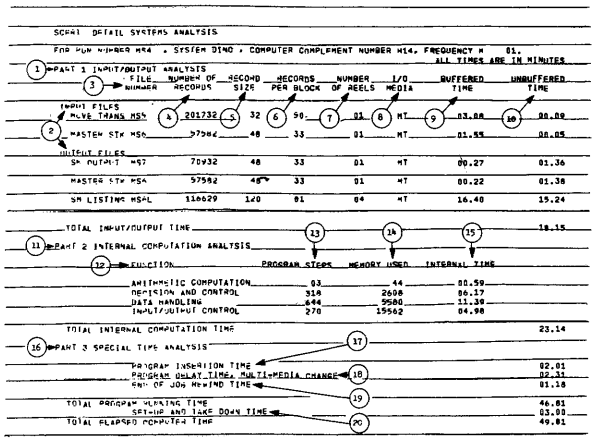


Figure 13. Detailed Systems Analysis.

analysis of internal computation, including time, memory, and program steps by function; i.e., arithmetic computations, decision and control, data handling and input/output control. Part III, a Special Time Analysis, forecasts the time required for program insertion time, program delay time due to errors, end of job rewind time, and other special functions. Finally, all timing forecasts are totaled to a net program running time. This report, in addition to providing detailed documentation and back-up for all other SCERT reports, can be invaluable to an analyst or programmer when designing the running program. It will solve such programming problems as input/output channel assignments, and the determination of optimum record size, blocking factor and program segment size.

COMPUTER COMPLEMENT REPORT

Report Explanation

This report reflects descriptive information for each computer configuration to be evaluated.

Column Explanation:

1 MODEL NUMBER—Identifies each component in the configuration.

2 MANUFACTURER—Identifies manufacturer of each component.

3 QUANTITY IN SYSTEM—The component quantity in each configuration.

4 PURCHASE PRICE—For each component and associated special features.

5 MINIMUM MONTHLY RENTAL—The minimum rental option for each component.

6 FLOORING SQ. FT.—Represents the minimum number of square feet of physical and work areas requiring false flooring for each component.

7 COOLING BTU/HR.—Shows BTU's required for cooling each component.

8 POWER SUPPLY—Reflects power requirements for each component.

9 CENTRAL PROCESSING UNIT—A subdivision of this report which shows all equipment connected on-line to the main frame.

10 PERIPHERAL DEVICES—The name of each component shown for the configuration.

11 SPECIAL FEATURES—Reflects all special features and control devices. All costs and other factors are shown with the appropriate device.

12 SATELLITE COMPUTER—Reflects the satellite computer (if required) to support the central processor.

13 TOTAL EDP SYSTEM—Reflects totals for all components for purchase price, monthly rental, floor requirements, BTU's and power in KW and KVA.

CENTRAL PROCESSOR UTILIZATION REPORT

Report Explanation

This report reflects analytical information computed by SCERT on each program run in the entire system for each computer configuration evaluated.

Column Explanation:

1 PERIOD—Identifies the periodic occurrence of the computer run.

2 FREQUENCY—Identifies the frequency of occurrence within the period.

3 RUN NUMBER—An identification number assigned by the analyst.

4 RUNNING TIME—This is the primary projection developed by SCERT simulation and represents elapsed running time of the program. It is based on optimum utilization of memory, instruction power and simultaneity.

5 SET-UP—Reflects net amount of set-up and take-down time required.

6 TOTAL TIME—A summation of running time and set-up time.

7 MEMORY CHARACTERS UTILIZED—Represents total requirements for each program, including instruction, work areas, I/O data, and software routines. An asterisk indicates memory has been exceeded, and SCERT has segmented.

8 PARALLEL PROCESSING—When equipment has the capability of performing parallel processing; SCERT calculates the effective parallel conversion time available. This figure is computed within the parameters of memory, operational time and I/O channels available during each run.

9 RUNS TOTAL—A summation of the SCERT projections of running time for all runs occurring in the period indicated.

10 AVERAGE TIME—The time required for all runs plus a proration of the running time for runs occurring more or less frequently than the period indicated.

11 NORMAL MONTH TOTAL—A summation of the total time required for all computer runs occurring monthly or more frequently.

12 PEAK MONTH TOTAL—The normal month time plus the time required if all quarterly, semiannual and annual runs were to occur during one month.

SATELLITE COMPUTER AND AUXILIARY EQUIPMENT UTILIZATION REPORT

Report Explanation

This report reflects analytical information computed by SCERT on each program run in the entire system for each computer configuration evaluated.

Column Explanation:

1 PROGRAM RUN NUMBER—An identification number assigned by the analyst.

2 SATELLITE COMPUTER—A projection of the hours of utilization of the satellite computer to support each program run. In this example, off-line equipment was used for normal data conversion operations, consequently, no utilization is reflected for a satellite computer.

3 OFF-LINE EQUIPMENT UTILIZATION—During the SCERT simulation, a pre and post history is developed for each computer run. SCERT analyzes this history to determine the off-line requirements for converting source document data to a media acceptable by the configuration being evaluated. It also determines the post run requirements for such off-line functions as printing, punching, etc. This off-line utilization is reflected in terms of hours per average month by type of function for each run, such as tape to print and card to tape.

4 DATA PREPARATION—Machine time requirements for all necessary data preparation such as key punching and paper tape punching for each program run are computed and reflected in terms of the hourly requirements per average month. In this example, source documents were not defined, therefore, data preparation time was not computed.

PROGRAMMING REQUIREMENTS REPORT

Report Explanation

This report reflects analytical information computed by SCERT on each program run in the entire system for each computer configuration evaluated.

Column Explanation:

1 RUN NUMBER—An identification number assigned by the analyst.

2 PROGRAMMED BY USER—A projection of the number of program steps which must be programmed by the user's staff to prepare each program.

3 SUB-ROUTINES—A quantitative reflection of the number of program steps which will be incorporated into each program run as a result of the utilization of manufacturer furnished sub-routines.

4 UTILITY ROUTINES—A quantitative reflection of the number of program steps in the manufacturer furnished utility routines used in each run.

5 PROGRAMMING EFFORT IN MAN MONTHS—A quantitative projection of the number of man months required to write each program run. This projection, although it may be no closer than plus or minus 15% of actual program effort required, because of the technique and the factors used in arriving at it, represents the best possible projection that can be achieved. The factors used for arriving at this projection are the number of programmers and their experience levels, the complexity of the computer run, and the language to be used.

APPLICATION SUMMARY REPORT

Report Explanation

This report reflects the summarization of information computed by SCERT at the program run level for each computer configuration evaluated.

Column Explanation:

1 APPLICATION CODE—The first character of program run number will be assigned by the analyst to identify each application in the system.

2 PROGRAM TIME—A projection of the CPU utilization hours for each application for an average month.

3 SET-UP TIME—Reflects net set-up and take-down time required.

4 SATELLITE COMPUTER UTILIZATION—A projection of the utilization

hours in support of the central processor for each application for an average month.

5 PROGRAMMING EFFORT IN MAN MONTHS—A summarization of programming manpower effort required for implementing all of the computer runs in each application.

6 PROGRAMMING EFFORT DOLLARS COST—An estimation of programming costs, based on an extension of the number of man months of effort multiplied by the average monthly salary of a programmer, as furnished by the user through the Environment Definitions.

COST SUMMARY REPORT

Report Explanation

This report reflects the SCERT projections for all costs associated with the operation of the entire system for each computer configuration evaluated for an average month.

Column Explanation:

1 BASIC HOURS—Represents the number of hours in the rental option selected by SCERT. SCERT analyzes, in terms of the number of hours of component utilization, all options offered and then selects the optimum one.

2 EXTRA USE—The number of hours of extra use required beyond the basic hours provided in the option selected by SCERT.

3 MONTHLY RENTAL—Represents the total basic hours plus extra use hours monthly rental for each component during an average month.

4 PURCHASE PRICE/XX—The purchase price of each component amortized over an XX month period. The period of amortization is selected by the user and defined to SCERT through the Environment Definitions.

5 MONTHLY MAINTENANCE—Represents the maintenance charges for each purchased component during an average month.

6 MONTHLY COST—A total of the amortized purchase price plus the monthly maintenance cost. This is a figure which can be compared to the monthly rental to assist in determining purchase versus rental.

7 DATA PREPARATION—A dollar figure representing the personnel costs required to prepare input data on a recurring basis to achieve the operation of the total system. This projection is based on factors furnished through the Environment Definitions, such as number of data preparation personnel and their mean salary, the user's estimate of their data preparation production, i.e., key strokes per hour; and the computations made by SCERT which reflect the total data preparation requirement for an average month.

8 EQUIPMENT OPERATION—A dollar figure representing the projection of equipment operation costs for an average month. This estimate is computed on data defined to SCERT, such as numbers and mean salaries of operation personnel; and the computations made by SCERT to determine the hours required for processing the entire system on the simulated configuration.

9 PROGRAM MAINTENANCE — An estimate of the user's personnel costs to maintain all computer runs in this system.

10 DATA MEDIA COSTS—Represents the costs computed by SCERT for unexpendable data media necessary for the implementation of this system. Examples are magnetic tapes or any other type of interchangeable data storage media.

DETAILED SYSTEMS ANALYSIS REPORT

Report Explanation

This report reflects detailed analytical information computed by SCERT on each program run in the entire system for each computer configuration evaluated.

Column *Explanation:*

1 PART I, INPUT/OUTPUT ANALYSIS—This part of the report is a complete analysis of all I/O functions in the system.

2 INPUT/OUTPUT FILES—A column showing file name.

3 FILE NUMBER — An identification number assigned by the analyst.

4 NUMBER OF RECORDS—The number of records for each of the files.

5 RECORD SIZE—The optimized record size computed by SCERT for each file.

6 RECORDS PER BLOCK—The optimum file block size computed by SCERT.

7 NUMBER OF REELS—The number of reels required for each tape file.

8 INPUT/OUTPUT MEDIA—A code specifying the type of media assigned to each I/O file. If the computer is a multi-channel machine, SCERT determines the channel assignment for each device and indicates it.

9 BUFFERED TIME—An estimate of that I/O time which can be buffered (or overlapped) by other operations during each program run.

10 UNBUFFERED TIME—A projection of that I/O time which cannot be overlapped and is therefore net time during each program run.

11 PART II, INTERNAL COMPUTATION ANALYSIS—During the simulation of each program run, SCERT compiles by functional category the number of program steps, associated memory utilized by work areas, and required time for all internal computation.

12 FUNCTION—Based on the internal computer activity codes specified by the analyst, SCERT determines a distributional breakout for the four functions of Arithmetic Computation, I/O Control, Data Handling, and Decision and Control.

13 PROGRAM STEPS—The number of instructions to be programmed by the user and to be furnished by manufacturer's software.

14 MEMORY USED—A total of the memory utilized for a function, including requirements for program steps, work areas, tables and I/O areas.

15 INTERNAL TIME—Time required for the computer to perform a function.

16 PART III, SPECIAL TIME ANALYSIS—An open ended analytical technique incorporated into SCERT to highlight for the program analyst the time consumed by those functions which are meaningful to the design of the system.

17 PROGRAM INSERTION TIME—A projection of the optimum time required to insert the program into memory based on available peripheral hardware. If SCERT determines that segmentation is required, the added time for inserting the various program segments is also computed and shown in this column.

18 PROGRAM DELAY TIME, MULTIMEDIA CHANGES—SCERT determines the number of reels required for tape files, and if during a program run an insufficient number of tape stations are available for each multi-reel file, it computes the time delay for rewind, take-down and set-up time for each reel.

19 END OF JOB REWIND TIME—The time required to rewind the longest reel.

20 SET-UP AND TAKE-DOWN TIME —The optimized time required to set up peripheral devices before each run and to take down completed work after the run. It is based on the number of operators and an analysis of the runs previous to and subsequent to each run.

A GENERAL-PURPOSE TIME-SHARING SYSTEM

Jules I. Schwartz, Edward G. Coffman, and Clark Weissman
System Development Corporation
Santa Monica, California

INTRODUCTION

Since June 1963, a Time-Sharing System has been operational at the System Development Corporation in Santa Monica. This system was produced under the sponsorship of ARPA and has utilized ideas developed at both Massachusetts Institute of Technology[3,4] and Bolt, Beranek, and Newman,[1,11] as well as some original techniques. Time-sharing, in this case, means the *simultaneous* access to a computer by a large number of independent (and/or related) users and programs. The system is also "general purpose," since there is essentially no restriction on the kind of program that it can accommodate. The system has been used for compiling and debugging programs, conducting research, performing calculations, conducting games, and executing on-line programs using both algebraic and list-processing languages.

This paper is divided into four major discussions. These are: (1) an outline of the capabilities provided for the user by the equipment and program system; (2) a description of the system's operation, with an analysis of the system scheduling techniques and properties; (3) a somewhat detailed description of two of the currently operating system service programs; and (4) a conclusion and summary.

CAPABILITIES FOR THE USER

Equipment Configuration

The major computer used by the Time-Sharing System (TSS) Executive is the AN/FSQ–32 (manufactured by IBM). Also used by the system is the PDP–1 (manufactured by Digital Equipment Corp.), which is the major input/output vehicle for the various remote devices.

The remote input/output devices available to users include Teletypes, displays, and other computers. These devices can be run from within SDC, and from the outside, with the exception of displays, which can be operated only a short distance from the computer. It is expected that computers to be used at remote stations will eventually include the CDC 160A, the DEC PDP–1, and the IBM 1410. (Currently only the 160A is being used, from an installation 400 miles distant from the Q–32.) Figure 1 is a description of the system's remote equipment configuration.

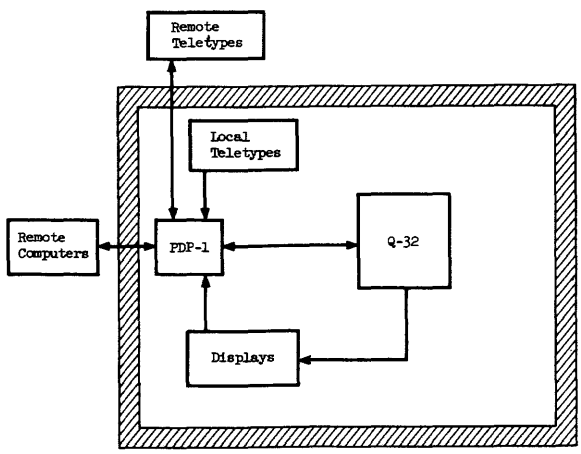


Figure 1. Remote Equipment Configuration.

The AN/FSQ–32 computer is a 1's-complement, 48-bit-word computer, with 65,536 words of high-speed (2.5 usec. cycle time minus overlap) memory available for programs, and an additional 16,384 words of high-speed memory available for data and input/output buffering; the latter memory is called input memory. The PDP-1 also has access to the input memory; thus, this memory serves as the interface between the two computers. In addition, the Q-32 has an extremely powerful instruction repertoire, including access to parts of words for loading, storing, and arithmetic; it also has an extensive interrupt system.

Figure 2 shows the principal components of the system and the important information-flow paths throughout the system. As implied in the figure, each main memory bank (16K words) is individually and independently accessible by three control units: the central processor unit, the high-speed control unit, and the low-speed control unit. High-speed I/O, low-speed I/O, and central processing can take place simultaneously out of different memory banks, or, with certain restrictions, out of the same memory bank. The high-speed and low-speed I/O operations originate, of course, from

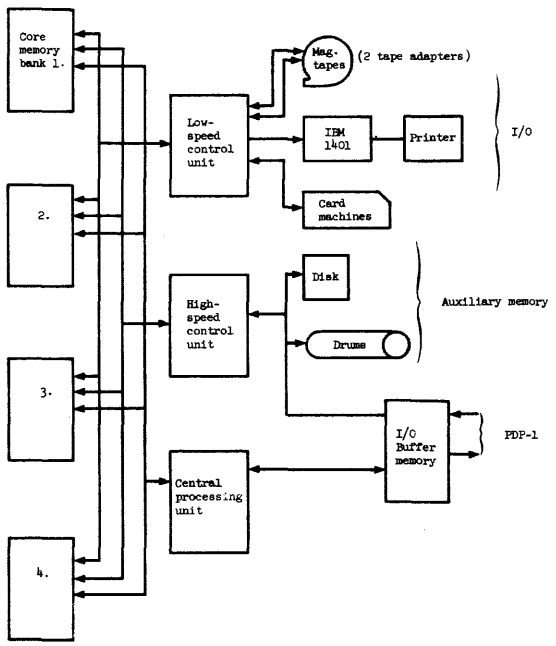


Figure 2. TSS Hardware System.

requests by the central processor unit. The low-speed control unit can service two or more low-speed I/O devices simultaneously, while the high-speed devices can only be operated individually, mainly because their cycle time approaches that of core memory.

A memory-protection mechanism and an interrupt, interval (quantum) clock (not shown in the figure) are also integral parts of the TSS computer system. On a bank-by-bank basis only, the memory protection mechanism provides the capability for inhibiting, under program control, the writing of information into one or more memory banks. The quantum clock has the following characteristics:

1. It can be set under program control to a time interval (quantum) anywhere in the range from a few msec. to 400 msec.
2. It can be made to interrupt computer operations after the set interval has elapsed, or after any power-of-two multiple of the set interval (up to a multiple of eight) has elapsed.
3. Under program control, it can be activated and reset.

A summary of the pertinent device characteristics is given in Table I below. The disk file shown in Figure 2 is currently being incorporated into the system.

The Time-Sharing System as it Looks to the User

The time-sharing user today communicates with the Time-Sharing System primarily by means of Teletype. He has at his disposal six basic commands to the system. Briefly, these commands are:

- LOGIN: The user is beginning a run. With this command he gives his identification and a "job number."
- LOAD: The user requests a program to be loaded (currently from tape, eventually, from disk). Once this command is executed, the program is an "object program" in the system.
- GO: The user starts the operation of an object program or restarts the operation of an object program that has been stopped. Once the user gives this com-

Table I. Characteristics of the AN/FSQ-32 Storage Devices

DEVICE	SIZE	WORD RATE	AVERAGE ACCESS TIME
Core Memory	65K	2.5 μsec./wd.	—
Inut/Output Core Memory	16K	2.5 μsec./wd.	—
Magnetic Drums	400K	2.75 μsec. /wd.	10 msec.
Disk File	4000K	11.75 μsec./wd.	225 msec.
Magnetic Tapes	16 Drives	128 μsec./wd. (High density)	5 to 30 msec. (no positioning), depending on whether the tape is at load point, and whether it is being read or written.

mand, he can send Teletype messages to either his object program or the Time-Sharing System.

- STOP: The user stops the operation of an object program.
- QUIT: The user has finished a particular job. Upon receipt of the QUIT, the Time-Sharing System punches a card with certain accounting information on it and removes the object program from the system.
- DIAL: The user may communicate with other users or the computer operators with this command.

In addition to these basic commands, the user has available to him a variety of on-line program debugging, or checkout, functions which give immediate access to any part(s) of an object program.

Briefly, these debugging functions include:

- Open: Displays the contents of the given memory or machine register and uses this as a base address for other debugging commands.
- Modify open register address: Changes the address of the opened register by the given increment or decrement.
- Insert: Inserts the given value into the opened register.
- Mask: Inserts values by the given mask.
- Mode: Displays values according to specified mode (floating, decimal, octal, Hollerith).
- Break point: When a specified point in the program is reached, notifies the user, and (on options) displays registers, and stops or continues the program. As many as five break points are allowed simultaneously.
- Dump: Dumps a given set of registers, either on Teletype or tape.

The actual commands to perform these functions usually include a symbol or address with one or two unique Teletype characters.

Additional Facilities Available to System Users

The commands and devices mentioned so far are facilities available to users or users' object programs directly through the Time-Sharing System's Executive. With these facilities one could run and debug programs that exist in a binary form. To make the system more useful, however, a number of additional devices (called service routines) are available to users. These are themselves run as object programs, so it is clear that there is no limit to the number of service routines that can eventually be made available.

These service routines include programs to file and update symbolic information; compilers; a fancy desk calculator; tape-handling routines; and a number of others including some advanced routines utilizing interpretive techniques. A more detailed description of these interpretive routines appears in the time-sharing applications section, below.

SYSTEM OPERATION AND TIME AND SPACE ALLOCATION

System Operation

The discussion so far has been primarily on the operation of the system from the user's point of view. The following is an over-all description of the system and how it operates.

Basically, the system operates as follows: All object programs are stored on drum, put there as a result of the LOAD command. When a program's time to operate arrives, or, preferably, ahead of this time, it is brought into high-speed memory. If bringing a program into its area in memory causes a storage conflict with another program, the latter must be restored to its place on drums (a process called swapping). A program's turn will end when it initiates an input or output request, when a machine or program error is detected, or when its time is up, the time allotted being determined prior to its turn. At the completion of its turn, its machine environment (e.g., accumulator, index-registers, etc.) is saved, and it either resides in memory until its next turn or is written on drums. This mechanism is controlled by the time-sharing Executive.

As stated before, there is no restriction on the type of object program that can run in the system. Therefore, as much input/output equipment as possible is made available to object programs; thus, object programs may use tapes, displays, and Teletypes for input and output. Other computers can also be treated as input/output devices; further, disk storage is available to object programs. Since it is impractical, in such a system, to have specific Teletypes or tapes referred to by object programs, input/output is done in a general fashion, with all input/output devices given arbitrary names by the object programs and declared to be files used by the object program during its run. Thus, only the Time-Sharing System knows what physical tape drivers, Teletypes, or areas of drums are being used.

The Time-Sharing System's Q-32 Executive occupies 16,384 words of memory, leaving the remainder of memory for object programs. The Executive that exists in the PDP-1 is primarily concerned with maintaining the flow of information to and from the remote devices. It does relatively little decision-making. However, it does determine the kind of input/output device concerned, the type of conversion necessary (if any), and the particular channel of the device with which it is communicating.

The time-sharing Executive in the Q-32 has eight major components. These include routines that perform input/output, perform on-line debugging, interpret commands, assign storage, and schedule object programs. By far the most distinctive feature of the time-sharing Executive, compared to other monitors or executive systems, is the scheduler. Accordingly, a more detailed description of time and space scheduling follows.

Time Allocation and User Capacity

The first problem considered in the Time-Sharing System (TSS) scheduling design was the determination of the minimum amount of time to be given each program during a response cycle of the system. A response cycle is that period of time during which all active programs (i.e., programs requiring central processing time) are serviced. Clearly, to satisfy TSS objectives, this quantum of time (q) must be at least as great as the average amount of time required by an object program to produce a response. Here, of course, we refer only to those programs designed to communicate with a user station (display or keyboard device), and to those programs for which a fast response is desired and can reasonably be expected. In other words, a user requesting a matrix inversion will (and must) expect to wait considerably longer than a user wishing only to see the contents of some register in his program.

Initially, it was obviously not possible to determine *a priori* the distribution of object-program operating times, nor was it even possible to define or classify the group of users requiring these data. The currently available information regarding user programs, and, to some extent, the experience of others, indicated that a q of 50 msec. was sufficient. The extensive recording now being performed during TSS operation is accumulating data that will much more accurately indicate the necessary q size.

In the following section, "worst-case" situations are being treated. "Worst-case" situations are being treated because they, by definition, give the overload threshold or capacity of the system; because they simplify the problem of having to cope with the distributions of object-program sizes and operating times; and because TSS will be operating at, or near, capacity for a high percentage of the time, if the present rate of usage continues. In some cases the "worst-case" values that are used had to be estimated. There is considerable evidence, however, to support the estimates given in the following approximation of the maximum number (n_{max}) of active users that can be serviced in one response cycle, when given the size of the response interval (t_r), the quantum size, and the hardware constraints.

In the current version of TSS the "worst-case" response cycle consists of the following recurrent, non-overlapping sequence of operations: dumping of the last program operated; loading of the next program to operate; allocation of the time interval for operation. For the values of q and t_r that are of interest, the number of active programs in the system can be much larger than the number of memory-protected programs that can be held in core memory at one time; therefore, the above sequence will virtually always be necessary for the operation of each program.

Assuming (as is presently the case) that object programs are not relocatable, we have (in view of the regular, cyclic operation of TSS) the following simple relation,

$$n_{max} = \frac{t_r\,(1 - \eta)}{2t_s + q} \qquad (1)$$

where t_s represents an average value for the time it takes to transfer a program from drum storage to core memory or vice versa, and η is the fraction of time (overhead) used by the Executive during each response cycle.

The fraction of overhead (η) is a difficult quantity to evaluate, and it depends to some extent on n_{max}. Because of the complexity of TSS operation, it is also difficult to estimate η through recording during TSS operation. From experience to date with the system, it is estimated that π ranges from two per cent to fifteen or twenty per cent depending on existing circumstances.

Equation (1) shows that, without major revisions in hardware, a significant inprovement in n_{max} can be achieved only through a decrease in the quantity $(2t_s + q)$. In particular, if object programs can be made dynamically relocatable, this quantity can be reduced to the value of $2t_s$ alone. Clearly, this is the best one can do, simply because the speed of the high-speed I/O section in swapping programs in an uninterrupted sequence represents a fundamental upper bound on TSS capacity. Further improvement necessitates an extensive increase in core-memory size, so that at least some active programs can remain in memory during consecutive response cycles. An increase in n_{max} brought about by an increase in the speed of the high-speed I/O section is not economically feasible as can be seen from the equipment description given earlier.

Assuming dynamic relocatability of programs, equation (1) changes to:

$$n_{max} \leq (1 - \eta)\,t_r/2t_s \qquad (2)$$

In practice, the extent to which the optimum is attained depends on the distributions of object-program sizes and operating times. If $2t_s$ is substantially larger than q, Equation (2) can, for all practical purposes, be considered an equality. Relocatability at load time would, of course, also significantly increase n_{max}, but the improvement that could be expected would be substantially less than that given in Equation (2). For a more specific evaluation of the improvement, a knowledge of the distributions just mentioned is necessary.

The linear relationships between n_{max} and t_r given by Equations (1) and (2) are shown graphically in Figure 3 with the following parameter values: q = 50 msec., η = 0.20, and

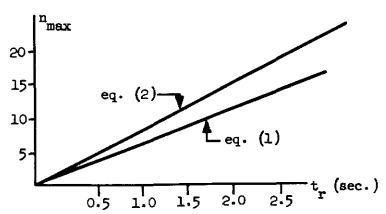


Figure 3. Response Time vs. Number of Users.

$2t_s$ = 100 msec. (corresponding to a program size of 16K words). At present, a value of 2.0 sec. is being used for t_r, with a resulting n_{max} of about 11.

Up to now, the efficiency of central processor utilization (relative to unoverlapped I/O time) has been considered of secondary importance, providing that user requirements have been met. Admittedly, this computational efficiency[3] is rather low in the "worst-case" situations. As will be seen in the next section, however, the way in which object programs are sequenced tends to maximize this efficiency for any given load situation. Clearly, the installation of dynamic relocatability in the system would allow an efficiency up to 50% since q can be made equal to 2t, without affecting Equation (2).

It should be emphasized that n_{max} does not represent the maximum number (N_{max}) of user *stations* that can be active at one time; it represents only the maximum number of user *programs* that can be serviced in a fixed response interval under the assumptions given earlier. It has been conservatively estimated that the associated object program is in need of central processor time only ten to twenty per cent of the time during which a user station is in use. Accordingly, it may be possible to make N_{max} considerably larger than n_{max} without significantly jeopardizing user-response requirements. Three important factors figure in the estimate of $\rho = n_{max}/N_{max}$:

1. Relative to computer processing speeds, many applications (e.g., debugging, gaming) consume considerable user time in thinking and output analysis.
2. The average user is less than professional in his use of input devices. A slow manual-input rate, coupled with occasional typing or format errors, will certainly tend to make ρ small.
3. Generally, computer output to user stations takes as much as one to ten seconds.

The estimate of ρ given above was based on the observation of these three factors during system operation and has been justified by the results of the limited amount of recording currently available. In obtaining the precise distribution of the quantity ρ it will be possible to determine the probability of overload for a given N_{max}, or to determine the N_{max} necessary for a given probability of overload. It should also be pointed out that, ultimately in TSS, as in a telephone exchange, several more user stations may be allowed than can actually be in use at one time. The extent to which N_{max} can be exceeded must again be determined by a distribution obtained in the same manner as for ρ.

Sequencing and Priorities

The sequence in which object programs are allocated time is determined by a priority scheme that favors the smaller programs that do not use low-speed I/O time. The amount of time allocated is given by the total time available (t_r), divided by the current number (n) of object programs requesting central processing time. When $n = n_{max}$, the time allocated is given by the minimum quantum discussed in the previous paragraphs.

The priority scheme was adopted to prevent low-speed I/O that was initiated by object programs from degrading the response of those users not using low-speed I/O. Users whose programs require low-speed I/O must expect poorer response, not only because of the low-speed operations, but also because of possible conflicts in object-program I/O requests. Each object program in the system receives a priority according to the criteria in Table II.

Table II. Priority Criteria

PRIORITY	PROGRAM CHARACTERISTICS
1	Program is less than 16K and does not use low-speed I/O.
2	Program is less than 32K and uses low-speed I/O or, program is between 16K and 32K and no low-speed I/O.
3	Program in excess of 32K.

During any given interval of time, Priority 1 programs will receive service first; Priority 2, second; and Priority 3, last. To prevent degradation of response by low-speed I/O, main

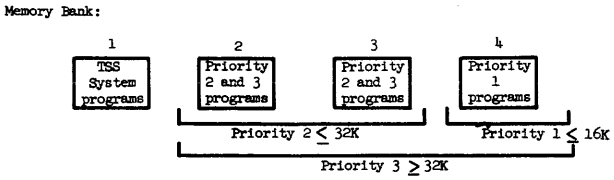


Figure 4. Main Memory Allocation.

memory is partitioned and allocated as shown in Figure 4. Because of the relatively small number of current TSS users, this storage allocation procedure has not yet been imposed on object programs. In the future when the number of Priority 2 and 3 users begins to cause a significant degradation in Priority 1 response, this scheme will be fully implemented as described.

Figure 4 shows that Priority 1 and 2 programs can be multiplexed, but Priority 3 programs preempt practically the entire machine. The priority scheme cannot solve the problem arising when a Priority 3 program undertakes a lengthy, low-speed I/O transfer. The majority of programs using low-speed I/O, however, concern tape transfers, which involve no searching, that take from 50 to 75 msec.

When a program completes operation prior to the expiration of its time allocation for any of the reasons given in the second paragraph of this section, the remaining time will be redistributed among the remaining users requesting service. As a result, the large Priority 2 and 3 users will generally receive more time than the Priority 1 users, thus increasing the potential utilization of central-processor time.

Space Allocation

Although the timing and speed limitations on TSS capability have been of concern, storage limitations are presently far more severe. Storage limitations can be largely removed, however, by acquisition of additional drum space up to the maximum of about 600K. Figure 5 gives a rough idea of how much drum storage must be provided for object programs, to achieve a balance between the speed and capacity of the system. The curves are obtained by letting $n_{max} = \rho N_{max}$ in Eq. (1), $t_r = 2.0$ sec., $\eta = 0.2$, and $\rho = 0.2$.

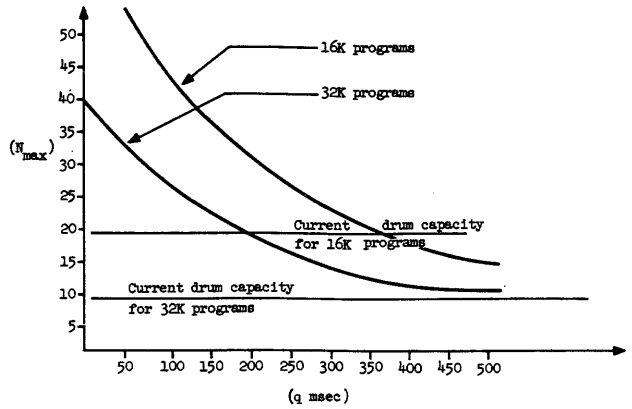


Figure 5. $N_{max} = \frac{1-\eta}{\rho} \cdot \frac{t_r}{2t_s + q}$.

In the initial TSS model (with only eight TTY users), auxiliary memory drum storage was partitioned and allocated in a fixed manner to provide an early working model of the system. This technique proved quite satisfactory at the time, but the number of input stations has now increased to about 48. To accommodate the additional users, a more efficient use of drum storage was necessary. The present method meets this requirement by allocating storage in a contiguous, "head-to-tail" fashion. The adapatability of this storage-allocation method requires searching an inventory of available drum space each time a new program enters the system, and periodically redistributing drum space to maximize the available amount of contiguous drum space. A possible disadvantage of this method is the additional overhead produced, especially when programs must be reshuffled to allocate a sufficient amount of contiguous drum space for a new program. Here again, the performance of this storage allocation technique must be evaluated by statistical recording, since the performance depends strongly on the distribution of program sizes, and on the rates at which programs enter and leave the system. However, at present (and in the foreseeable future) the above rates are so low that the additional overhead produced is negligible.

Future Improvements in TSS Scheduling

There are many ways, including both hardware and software additions to the system, in which the capacity and scheduling efficiency of

TSS can be enhanced. The more or less obvious hardware improvements include:

- Additional core memory
- Additional drum memory
- Relocation mechanism
- Disk storage

The effects of a relocation mechanism and additional drum storage have been described in the previous sections. Additional main memory can be expected to allow for a larger Executive system, larger object programs, and greater scheduling efficiency. However, a substantial improvement in scheduling efficiency must be predicated on the existence of a relocation mechanism, when one makes the obvious assumption that the memory size is small compared to the total size of all active object programs.

The disk file, which is just now being installed and checked out, will supplant tapes for all those applications in which disks are faster and use less machine time. It is expected that disks will be used to store a program library for TSS usage and to store large data bases for object program usage. It is conceivable that disks will also be used for program swapping. The first use of the disk file promises to eliminate a high percentage of manual operations associated with program loading from tapes, and thus to reduce greatly the corresponding delay experienced by users. The second use should save considerable time for the user whose application involves searching through large data bases.

Although the estimates given in this paper are based realistically on current experience, it is not unlikely that user characteristics will evolve quite differently than predicted. Program sizes and/or operating times may grow to a point that invalidates the "worst-case" figures given in this section. It is possible, however, to counteract a certain amount of this degradation by certain improvements in scheduling logic. One improvement would be obtained by taking advantage of the fact that a fairly large class of users exists for whom responses substantially greater than one or two seconds are quite acceptable. In short, it is possible to assign response levels to each user and to service each user just frequently enough to ensure his level of response. Furthermore, the disk file can be used for swapping those programs for which short responses are not necessary. Provided that disk access is in parallel with other high-speed I/O activities, the effective swapping speed can retain the same order of magnitude as for drums.

There are many programs that do not alter themselves during their execution. Thus, as another software improvement, these programs could be treated by the system in two sections: an instruction section and an environment (data) section. During a program's execution it would never be necessary to write the instruction section back on drums; only the environment section and the machine conditions at interrupt would be written back on drums. These and other improvements to TSS are under present investigation. Of principal concern in the investigation of these system changes is the amount of overhead they produce. In some cases the increase in overhead exceeds the expected "improvement" in operating speed and efficiency.

TIME-SHARING APPLICATIONS

To illustrate the "general purpose" nature of the Time-Sharing System, we focus on two interesting programming systems currently operating on TSS as service systems for the user. The first, IPL-TS, is a complete list-processing system for the Information Processing Language V developed by Newell, Simon, and Shaw.[12] The second, TINT, is an on-line *T*eletype *INT*erpreter for the JOVIAL algebraic language developed by SDC.[16, 17] When the Time-Sharing System is equipped with these two programming-language systems, the user is immediately provided with a familiar programming system to ease his transition to programming for time-sharing, and allowed to use, with little or no modification, any code he may have previously written in IPL-V or JOVIAL for other machine systems.

IPL-TS Description

IPL-TS executes interpretively IPL-V code written in accordance with the latest published IPL-V conventions.[13] Some exceptions are noted, particularly in the IPL-TS I/O conven-

tions dictated by machine limitations and time-sharing procedures. More significant, however, are the extensions provided by IPL-TS in the areas of mode of code execution, and improved on-line communication.

IPL-TS can operate in one of two modes at the programmer's option: the "production" mode or the "debugging" mode. The production mode is designed for maximum code execution, and is used essentially for checked out code.

Though code is still executed interpretively, a suppression of all debugging functions in the production mode has produced a four-fold increase in execution rate over debugging-mode operation. Execution rates of over 400,000 cycles per minute, which compare favorably with other non-time-sharing IPL systems, are common. To the IPL-TS user, production-mode operation is analogous, as we shall see later, to the TINT user compiling his checked-out code with the Time-Sharing JOVIAL Compiler (JTS). Debugging-mode operation, on the other hand, is designed for maximum user efficiency and greater on-line programmer control over the execution of his program. The debugging mode allows all the standard IPL options; it also permits a number of on-line functions not common to IPL systems. These include:

1. Optional breakpoint action at any monitor point, whereby the currently executing program is suspended until completion of the execution of any on-line, programmer-specified routine;
2. On-line, symbolic program composition and/or debugging;
3. Optional automatic or on-line programmer-controlled execution of a full "back trace" routine that prints up to the last 100 interpretation cycles. This routine is executed by IPL-TS automatically at each system-detected error occurrence as a debugging diagnostic; and
4. A flexible, "thin skinned" system error trap mechanism permitting programmer specification of trapping actions for all system-detected errors.

TINT Description

TINT is a two-pass interpretive program system for time-sharing use, and operates upon a subset of the JOVIAL problem-oriented language.

TINT includes a generator, a set of operator subroutines, and the interpreter. The generator was acquired from a current IBM 7090 JOVIAL compiler and was modified to handle the particular JOVIAL dialect used by TINT. The operator subroutines and the interpreter are original code developed specifically for TINT.

The generator (first pass) scans the input JOVIAL statements and translates them into an intermediate Polish prefix language. Grammer checking is performed during the translation. The language subset allowed may include the arithmetic, relational, and Boolean operators; procedure calls; data table, array, and item (integer, floating point, and Hollerith) declarations; and the GOTO, IF, STOP, READ, and PRINT statements. The READ and PRINT statements were added to the language specifically for time-sharing operation.

Operator subroutines comprise the primitive functions used by the interpreter to perform the actions specified in the intermediate language. The interpreter (second pass) scans the intermediate language for the current operator prefix and its arguments, and executes the corresponding operator subroutine that computes on these arguments.

The TINT user is permitted a number of options in composing and executing his code. He may reference code stored in a binary library tape of his own composition; he may file away any current code on tape for subsequent use, or for compilation with JTS after the code has been exercised and debugged; and, he may optionally execute code from a prestored tape or from the Teletype.

On-Line Program Composition

Both IPL-TS and TINT allow the user to write symbolic programs on-line and to execute them immediately, by themselves or in conjunction with previously coded routines. With IPL-TS, the programmer uses the special system routine, Linear IPL (LIPL),* which accepts

* LIPL was designed and coded by R. Dupchak while consultant to the RAND Corporation, Santa Monica.

```
LIPL READY!

RT ACKTEST=(A0 A1 A2 A3,ACKTEST)
   A0=(J154 10$CYC.D$ J157 10$H3=$ J157 10+10  J161
     10$SEC.D$ J157 10$TIME=$ J157 10+10 J161
     10$ACK($ J157 10M0 J157 10$,$ J157 10N0 J157 10$)=$ J157)

* A0 SETS UP PRINT LINE *

   A1=(10H3 J120 J50 *$1)
* A1 SAVES H3 COPY IN W0, AND CLOCK IN $300 *

   A2=(1MM0 J117 10N0 709-1 J125,JK) 9-1=(J117 709-2 10N0 J125,9-3)
  9-2=(10+1 10N0 J111 A2,9-3 50K1 10M0 10M0 J111 A2 J125,JK)

* A2 COMPUTES AC(M0,N0) *

   A3=(10+9 J160 10H3 11W0 11W0 J111 J157 10+29 J160 *$3 10+49
     J160 10N0 J157 J155 11W0 J9,J30)

* A3 COMPUTES DELTA  H3 AND DELTA TIME. ALSO SETS VARIABLES IN
 PRINT LINE AND PRINTS LINE *

   M0=+2 N0=+1 K1=+1  * INITIAL VALUES * NL  GT ACKTEST
CYC.DH3=178          SEC.DTIME=0.2407    ACK(2,1)=5
CYC.DH3=1102         SEC.DTIME=1.7929    ACK(2,5)=13
CYC.DH3=5254         SEC.DTIME=3.4091    ACK(2,13)=29
CYC.DH3=25294        SEC.DTIME=16.3442   ACK(2,29)=61
CYC.DH3=190687       SEC.DTIME=121.8779  ACK(2,61)=125

!STOP
```

Figure 6. Typescript of Ackermann's Function.

IPL code on-line in a symbolic, linear, parenthesis format convenient for keyboard input. Figure 6 presents an example of LIPL being used to compose and execute Ackermann's function[8] on-line. TINT, which was developed specifically for on-line program composition, accepts JOVIAL statements on-line in the same linear format used for compiler input.

The ability to program on-line frees the programmer from having to concern himself with all the formalities of punched card accounting. With experience and facility, he programs on-line directly from his thoughts or, for more difficult problems, directly from a flow diagram, circumventing such time-consuming tasks as program-coding-sheet preparation, key punching, card sorting, editing, and prestoring. The time saved by the programmer can be applied to other coding tasks or to quality review of his current code.

No programmer, of course, could compose a large program at one sitting with either of these systems, but this is a human, not a system, limitation; LIPL has no upper bound, and TINT's 600-statement limit effectively exceeds a human's short-term comprehension. Optimally, these systems should be used for programs that can be written and debugged in one or two sittings (usually under 100 IPL instructions or 50 JOVIAL statements).

There are three immediate consequences of this practical size limitation. First, many nontrivial, one-shot programs, such as for statistical computations, can be coded, debugged, and executed at one sitting. Often a programmer himself will refrain from writing such programs, knowing the time and effort involved. Figure 7 shows the Teletype communication resulting from an exercise using TINT as a "desk calculator" for computing the standard deviation of a set of research data. Second, large programs take on a modular structure; that is, large programs become a concatenation of numerous smaller programs and subroutines. Third, programmers begin to amass personal libraries of short utility subroutines, which they use to build larger programs. Clearly, consequences two and three would not exist, except in trivial cases, if it were not possible to work one day with code developed on prior days. Both IPL-TS and TINT provide this capability.

```
LOGIN 0173 JDX.25
$OK LOG ON 14
LOAD TINT 1796
$WAIT

$LOAD OK
GO
 $MSG IN

          START   "BEGINS NEW PROGRAM"
         ITEM N F $   "NUMBER OF CASES"
         ITEM SUMX F $    "SUM OF VALUES"
         ITEM XBAR F $    "ARITH. MEAN"
         ITEM SDEV F $    "STD. DEVIATION"
         READ  N, SUMX $
         XBAR = SUMX/N $
         PRINT 6H(MEAN =), XBAR $
         SDEV = ((SUMX**2.0-N*XBAR**2.0)/(N-1))**0.5 $
         PRINT 8H(S. D. = ), SDEV $
         TERM $    "CAUSES EXECUTION OF PROGRAM"

             N = ? 12.0

             SUMX = ? 1478.0

          MEAN =     123.2

          S. D. =      10.3

         ILT EXECUTION COMPLT

!QUIT
 $MSG IN
```

Figure 7. Example of the Use of TINT as a "Desk Calculator."

TINT may accept symbolic input from magnetic tape, and can integrate this input with on-line Teletype input when so directed by the user. Thus the results of one day's coding can be filed on tape for later use. An alternative, if the symbolic JOVIAL statements have been executed and debugged, is to compile the code and save the binary output on a binary library tape, thus, again, integrating previous work with current code; however, the binary library approach has greatest value when used for utility routines.

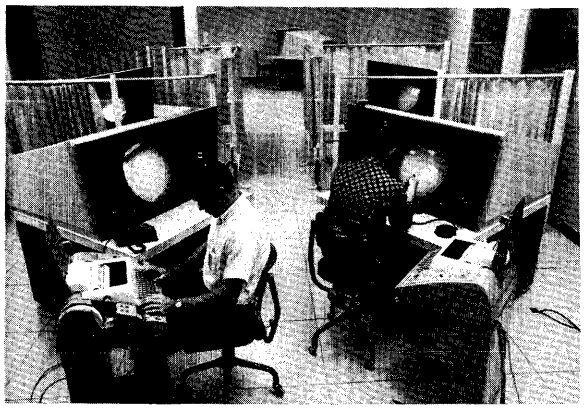


Figure 8. Accessing the Computer with Model 33 Teletypes and Displays.

IPL-V is essentially a language of subroutines (composed from an inventory of some 200 system subroutines called J routines or primitives). Programs written in IPL-V are usually modular hierarchies of subroutines. Therefore, on-line composition of IPL-V programs is a natural extension of the language, and many alternatives for continuity of programming across many days of operation already exist within the language. For example, the programmer may "fire" a J166 (Save For Restart) at any time and continue from that point at a later date, or he may load a program from symbolic tape using the loader or J165 (Load Routines and Data) and continue using LIPL on-line.

Therefore, the attributes of IPL–TS and TINT, when combined with a programmer's imagination and skill during on-line program composition, reduce significantly the tedious, uncreative tasks of code preparation and increase productivity. This point is particularly apparent to all programmers who have been required to debug code that they wrote several days earlier, and that has grown "stale" while it was being keypunched, compiled, and executed. Instead of expending additional time and energy becoming reacquainted with his code before he can correct his errors, the programmer can, by composing the code on-line and executing it immediately, debug while the code is still fresh in his mind.

On-Line Program Debugging

The particular ability of IPL-TS and TINT to detect, locate, and correct program errors on-line is perhaps their greatest asset, since it leads to substantial decrease in program turnaround time. In effect, IPL–TS and TINT increase the programmer's debugging efficiency by allowing him to check out more code per day than would be possible with non-time-sharing operation.

Error Detection is the first step in debugging any program. Errors may be classed as either grammatical errors in language or format, or logical errors in code execution. The generator screens out most grammatical errors for TINT, and either the loader or LIPL performs the same task for IPL–TS. Logical-error detection, however, is a more difficult task, even with IPL-TS and TINT. The advantage of these systems for error detection is their responsiveness to the programmer. He may choose to develop on-line, special-purpose debugging tools to suit his individual preference, or he may use those debugging tools provided by the system. For example, IPL-TS currently provides an error trap for some twenty illegal IPL operations resulting from faulty program logic; when such errors occur, IPL-TS attempts to provide the programmer with as much information as possible to help him correct his error. First, an error message is sent to the programmer to inform him of the error's occurrence and of its nature. Second, a special system routine, Trace Dump (discussed below), provides him with a "back trace" of the code leading up to the error to help him locate the cause of the error. Finally, the system pauses at a breakpoint, to allow him time to correct the error. However, all three steps may be altered, since the IPL-TS error trap mechanism is designed with a "thin skin" to allow the programmer to substitute his own trapping action in lieu of that provided by the system.

With TINT, logical-error detection is left more to the imagination of the programmer. TINT allows the programmer to insert a PRINT statement, with numerous item names as arguments, at any point in his program. When it encounters this statement during program execution, TINT responds by printing on the user's Teletype the current values of all specified items. In this fashion, the program-

mer may take item snapshots at critical points in his program. The power of the PRINT statement for logical-error detection is amplified when combined with the TINT READ statement. The READ statement is the converse of the PRINT statement. When TINT encounters this statement during program execution, the programmer must insert the current values of prespecified items. By judicious use of the READ and PRINT statements, the programmer can repeatedly exercise a program with different initial conditions and review his results with input/output transfer-function analysis.

Thus, on-line user-program communication increases a programmer's debugging efficiency by increasing his ability to detect program errors. It is typical for a programmer, checking out new code with IPL–TS or TINT, to detect and correct half a dozen program errors in the first hour of operation; such error correction might easily have required a week with conventional programming systems.

Error location, the pinpointing of the erroneous code, is often considered no different from error detection. This may be true for grammatical errors, but is far from true for logical errors. The knowledge that an error exists does not, in and of itself, narrow the search for the error's location. The user of IPL-TS, therefore, is provided with a description of the system-detected error and the aforementioned back trace of the code leading up to the error. Back tracing by the system is performed in the debugging mode by the special system routine Trace Dump, which prints a full trace of up to the last 100 interpretation cycles, in reverse order (last cycle first). The number of previous cycles printed is controllable on-line. Experience shows that the location of an error can usually be found within the first five cycles printed, and that it is rarely necessary to go deeper than ten cycles back. For logical errors not detected by the system, the programmer has available all the standard IPL-V Monitor Point functions; in addition, IPL-TS extends these functions to include breakpoint operation as a programmer-initiated option. The option may be invoked at load time or during program execution. In addition, the IPL primitive J7 (Halt) has been implemented as an alternative breakpoint mechanism. When a breakpoint is encountered by IPL-TS, the programmer is notified and requested to enter the name of any regionally defined routine, which is then executed immediately. Upon completion of the routine, the programmer is again queried. He may continue to fire routines at the breakpoint, or he may exit back to the prior program, the context of which has remained undisturbed.

Breakpoints are not a panacea for locating erroneous code; however, they do provide additional control flexibility at critical points in a program. In fact, the user of TINT must rely almost exclusively on breakpoint logic for locating erroneous code: the aforementioned READ and PRINT statements are in effect breakpoint statements. For elusive errors these statements may be used to bracket groups of JOVIAL statements, and in extreme cases, individual JOVIAL statements. TINT also provides a STOP statement, which is also a breakpoint statement. When the interpreter encounters the STOP statement, the program is suspended until directed by the user to continue. The user may also reexecute his program from a STOP breakpoint, or he may enter new code or edit prior code before continuing. TINT's STOP statement is analogous to the IPL-TS J7 (Halt) primitive.

Error correction in symbolic code with either IPL-TS or TINT is essentially on-line program composition. LIPL allows the IPL programmer to erase, extend, or modify selectively any user routine existing in the system. TINT, similarly, allows the programmer to edit any JOVIAL code written, on a statement-by-statement basis.

Here, again, the programmer's control over his program is effectively increased. He can correct code in several minutes instead of the several days typical with most computer installations.

SUMMARY AND CONCLUSION

There are some obvious advantages to this kind of system that have been borne out in practice. There is a large class of problems whose compute time is extremely small in relation to the total time the problem is on the computer. This is because a large percentage

of time is taken up by human thought and computer input/output. In fact, the use of a computer for this kind of application in a non-time-sharing mode is so inefficient that it would not be worthwhile to run. There are many examples of this kind of problem. The one that most programmers are familiar with is console debugging, that is, the checkout of programs with the programmer at the computer—anathema to most computer managers, but desired by a large number of programmers. These kinds of applications have been run with a high degree of success in this Time-Sharing System, with each person involved actually feeling he has the whole computer to himself.

At the other end of the spectrum are those programs that compute for essentially one hundred per cent of the time they are on the computer. If these programs compute for long periods, say a matter of minutes, they will completely usurp their allotted time and thus tend to make the on-line user wait for the maximum response period possible. Time-sharing does not benefit this kind of user, except that this kind of program can be run "in the background" while other on-line interaction programs are idle. In the SDC installation, the percentage of these long-period compute programs has been small, so that no serious system response time delays have been noticed from them.

Questions frequently asked are, "Do people like the system?" "Does it produce better results than other, more standard techniques?" Both the questions are difficult to answer in an absolute sense. However, some reasonable observations can be made that apply to this system and probably to others of this kind.

First, those on-line interaction programs that used to run in a non-time-sharing mode but were converted to time-sharing produce results that are as valid as before but with greater efficiency in computer operation, since a number of different ones are run simultaneously.

Next, the on-line debugging capability has proved very valuable. This system of debugging gives a feeling of closeness to the computer and control over the program, so that debugging time is reduced considerably while the efficiency of computer utilization stays high. Also, although the tools available so far have been relatively few and unsophisticated, one can see the advantages to be gained by giving everyone immediate access and response from a computer. "Directed" computer runs are the mode of operation. Every step taken is taken only as a result or verification of the previous step. If things do not go as planned, alternative paths can be followed immediately. Before time-sharing, one had two choices: "submitting" of a run, followed by an anxious waiting period climaxed by a sigh (or worse) and a resubmitting of the same run; or one-man on-line interaction with the computer, which benefitted that person, but caused consternation on the part of others waiting for computer runs.

This kind of system must be made foolproof. Due to the nature of this system, one must have a reasonably long time of uninterrupted operation to get satisfactory results. This implies several things:

1. The system Executive must be reliable and able to account for any condition that may arise, including object program and machine errors.
2. The machine must be reliable. Although the system must provide the ability to analyze each computer error and isolate and stop only the particular object program or programs affected, frequent or solid computer errors can cause the entire system and all object programs to terminate.
3. Certain hardware features are essential. These include: *Memory protection*—the ability to prevent object programs from destroying each other or the Executive system; and *high-speed large-storage random-access devices*—the major bottleneck in a system of this kind is the slow rate at which object programs can be moved in and out of memory. Also, the use of magnetic tapes for such functions as the permanent storage of programs and data files creates operational and timing problems that can be overcome with the use of large drums or disks; also essential is *clock interrupt capability*—the system requires that no single program run for an excessively long time.

Therefore a clock that can be set to interrupt operation at various intervals is necessary for complete control and the assurance of adequate response time.

When this Time-Sharing System first became operational, it had no memory protection, its Executive was unreliable, and its computer was beset by a much heavier load than it was used to and reacted accordingly. With these obstacles, the early users were subject to frustrations unlike many found in the twentieth century. The system's life expectancy was no more than ten minutes. The only remarkable thing about the early months was that anything useful was accomplished. Interestingly enough, however, some work was accomplished, primarily through patience on the part of the users. With the passage of time, many of the problems have been alleviated through both equipment and programming improvements, so that now the system runs with considerably more continuity and reliability.

Since the system became operational, it has been used in a wide variety of applications. These applications have, for the most part, been checked out using the Time-Sharing System and have been run productively during time-sharing. Some of the specific applications for which time-sharing has been used are:

- Natural Language Processors—used for parsing English sentences, answering questions, and interpreting sentence-structured commands.
- Group Interaction Studies—in which teams or players are matched against each other and the computer is used to measure individual and team performance.
- General Display Programming—in which the programs are used as vehicles for generating and modifying visual displays according to the users' keyboard inputs.
- A FORTRAN-to-JOVIAL Translator—symbolic JOVIAL program tapes are produced for FORTRAN tape inputs.
- Simulated Alternate Mobile Command Post—a realistic simulation of the A.M.C.P. has been produced, and the display requirements for this organization are studied within this framework.

Of course, a number of other routines, games, and services have been and are being developed under the system.

One of the "disadvantages" in using a time-sharing system such as this is the fact that most computer runs require the presence of one or more people. Users of many large-scale computers are accustomed to remaining detached from the actual computer runs and are sometimes reluctant to follow the runs closely. However, the elapsed time for completing jobs using these "on-line" techniques is normally dramatically reduced compared to a more remote operation, and this reduced time has been noted in the use of the time-sharing system.

It is interesting to watch a group of people using a computer simultaneously but solving different problems using different tools. At the computer console itself, one can usually see all the available tape drives busy, typewriters busy, drum indicators indicating the drums are busy, the punch punching, on occasion, and the card-reader going at anywhere from quarter to full speed. For those who judge the worth of a computer by the amount of equipment used per second, time-sharing is well worth its investment.

Since the system has been under development (it was begun in January 1963), the number of existing services has been expanding rapidly. One can envision the development of an increasing number of on-line programming aids and techniques of utilizing keyboards, displays, and groups of computers to make a time-sharing network a truly powerful device.

It is certainly conceivable that, in the not too distant future, many people will have at their fingertips a device that, at a reasonable cost, enables them to enter an operating network such as this one. While in this network, they will have access to routines, techniques, and computing power unavailable to them by other means. The computing power will include not only the Executive computer but the other computers that are in the network as well. Thus, the possibility of large-scale time-sharing networks seems to be one of the more promising developments in computer technology today.

BIBLIOGRAPHY

The following represents a collection of general as well as source material.

1. BOILEN, S. "How the Time-Sharing System Looks Now," Cambridge, Massachusetts, Bolt, Beranek and Newman, Inc. (Unpublished Memo. TS-3), April 2, 1962.
2. COFFMAN, E. G. *A General Flow Chart Description of the Time-Sharing System.* SDC TM-1639/000/00, December 12, 1963.
3. CORBATO, F. J., and others. *The Compatible Time-Sharing System. A Programmer's Guide.* Cambridge, Massachusetts, M.I.T. Press, 1963.
4. CORBATO, F. J., M. MERWIN-DAGGETT, and R. C. DALEY. "An Experimental Time-Sharing System," *Proceedings of the Spring Joint Computer Conference.* 1962, pp. 335-344.
5. FREDKIN, E. "The Time-Sharing of Computers," *Computers and Automation.* v. 12, November 1963, pp. 12-20.
6. GALLENSON, L. *On-Line I/O Processor for the Command Research Laboratory. The PDP-1-C-30.* SDC TM-1653, December 23, 1963.
7. KEMPER, D. A. *Operation of CRL Teletype System.* SDC TM 1488/000/00, September 18, 1963.
8. KLEENE, S. C. *Introduction to Metamathematics.* Van Nostrand, 1952.
9. LICKLIDER, J. C. R. "Man-Computer Symbiosis," *IRE Transactions on Human Factors in Electronics,* V.HFE-1, March 1960, pp. 4-10.
10. LICKLIDER, J. C. R., and W. E. CLARK. "On-Line Man-Computer Communication," *Proceedings of the Spring Joint Computer Conference,* 1962, pp. 113-128.
11. MCCARTHY, J., S. BOILEN, E. FREDKIN, and J. C. R. LICKLIDER. "A Time-Sharing Debugging System for a Small Computer," *Proceedings of the Spring Joint Computer Conference,* May 1963, pp. 51-57.
12. NEWELL, A. (Ed.) *Information Processing Language V Manual.* Englewood Cliffs, N.J., Prentice-Hall, Inc., 1961.
13. NEWELL, A. (Ed.) *IPL-V Programmer's Reference Manual.* RAND Corporation, RM-3739-RC, June 1963.
14. ROSENBERG, A. M. *Externally-Generated Priority Assignment for Program Operation in the ARPA-SDC Time-Sharing System.* SDC TM-1159/000/00, April 8, 1963.
15. ROSENBERG, A. M. (Ed.) *Command Research Laboratory User's Guide.* SDC TM-1354, November 19, 1963.
16. SHAW, C. J. "JOVIAL," *Datamation* 7, 6 (June 1961), pp. 28-32.
17. SHAW, C. J. "Programmer's Look at JOVIAL in an ALGOL Perspective," *Datamation* 7, 10 (Oct. 1961), pp. 46-50.
18. SLAYBAUGH, J. A. *The AN/FSQ-32. A Description and Coding Manual for Experienced Programmers.* SDC TM-1489/000/01, December 1963.
19. STRACHEY, C. "Time-Sharing in Large Fast Computers," *Proceedings of the International Conference on Information Processing,* Paris, UNESCO, 1960, pp. 336-341.
20. WEISSMAN, C., and M. KAHN. *IPL-TS Programmer's Reference Manual.* SDC TM-1581/000/00, December 16, 1963.

REMOTE COMPUTING—AN EXPERIMENTAL SYSTEM
Part 1: External Specifications

T. M. Dunn and J. H. Morrissey
Development Laboratory, Data Systems Division
IBM Corporation
New York, N. Y.

INTRODUCTION

Background

Remote computing has been around as long as computers themselves.[1] More recently, interest has revived in providing remote users with convenient, economical access to a large central computer. Considerable attention has been addressed to its economics[2] and practicality.[3] Several batch-oriented systems have been implemented.[4, 5] The techniques of time-sharing[6, 7] a large[8, 9, 10] or small[11] system have been described, as have the attendant advantages of man-machine interaction[12, 13] for symbolic mathematics[14] and program testing.[15] Several input-output devices have been considered, including typewriters,[16] displays,[17] and dial-voice equipment.[18]

The management,[19, 20, 21] systems analysis,[22] program testing,[23] and documentation[24] of specialized real-time systems have also been emphasized, but much less attention has been given to the design of general-purpose on-line systems.

This paper reviews some general system requirements and applications criteria leading to basic design objectives and constraints for remote-computing systems. An experimental system using a number of remote terminals time-sharing a standard computer is then described.

System Requirements

There are several requirements that must be considered when designing a practical remote-computing system.

1. The remote user does not have access to experts for programming assistance and advice. If he uses a problem-oriented language to express his problem, he requires that the request for and display of debugging data be consistent with this programming language.
2. Because his jobs are processed completely without human intervention, the remote user obviously cannot communicate his desires to a machine operator. This leads to several considerations:
   a. The command statements used to regulate the system should have a form and content consistent with the programming languages employed.
   b. The remote user requires a powerful command structure; he should have the ability to state such things as run time, job status, error procedures, and disposition of output.
   c. The conversational remote user requires access to many of the facilities available to the machine operator in the form of console buttons, lights, and switches. He should receive steady reassurance that "all is well" by some

form of periodic "blinking" at his terminal. He also needs the ability to stop his "machine" at any time and without loss of data—so that he can perform such simple functions as changing some printer paper, placing more input cards in a reader, or discontinuing a job.

3. The remote user is very conscious of input/output volumes. He must have the capability to modify decks without complete retransmission, and he should have the option to selectively inspect and list output data, as opposed to massively transmitting entire output files. Also in this spirit, he desires to keep his various decks in random storage—quickly and conveniently available for modification, processing, or review.
4. Finally, the remote user should be given the impression that he is the only user and that he is in complete control of the situation. More specifically, in a time-sharing environment, he should be totally secure from unwanted, possibly destructive, interaction by others. Ideally, the computing and response rate of his terminal should not radically fluctuate according to the demands of the rest of the user population; in other words, his "share" of the central system should perform at a relatively uniform processing rate.

Application Criteria

The following criteria were among those used in deciding whether commercial or scientific applications were more amenable to remote operation:

1. Time devoted to program development vs. production runs;
2. Importance of job turnaround vs. computer throughput;
3. Available programming languages;
4. Conversion problems;
5. Reliability objectives;
6. Input/output volumes;
7. Random-storage requirements.

It was concluded that there was more immediate technological significance and lower hardware-software risk in placing initial emphasis primarily on the scientific applications area.

Design Objectives

The following functional design objectives were then established:

1. Output data should be as user-oriented as the source language;
2. Diagnostic messages and logical analysis should be definitive enough to allow program debugging to take place at the same level as program construction;
3. The user should have immediate and sustained access to the computer;
4. The user should have the ability to execute, alter, and change values, variables, and formulas, and to request information selectively;
5. The system should be at least as easy to learn as the FORTRAN[25] language;
6. The print volume should be minimized without loss of quality, on demand of the user;
7. The system should provide the shortest possible solution time, ideally no longer than the time required to construct and run the solution itself.

Design Constraints

Finally, the following restrictions were imposed:

1. Use only an existing standard equipment configuration;
2. Use, and stay consistent with, an existing language.

The first constraint serves to keep attention primarily on fundamental programming problems and discourages the favorite desire of many engineers to solve systems problems by the design of a new feature or the development of new devices.

The second constraint serves to keep attention primarily on the processor design and discourages the favorite desire of most programmers to solve systems problems by the design of new languages or the development of new compilers.

The Approach

Our approach to accomplishing the objectives fuses the old technique of interpretive execution with the relatively new one of time-sharing a CPU. Thus the cost of sustained access to a computer by an individual is spread over a wide base. The internal form suitable for interpretive execution retains all the information contained in the user's original statement of the problem, thereby making symbolic debugging possible. Together, these two techniques make the conversational mode of operation on current equipment a practical reality.

Nevertheless, the service this system performs is not a matter of cleverly getting something for nothing, but a justifiable trade-off. Execution time is greater, but elapsed solution-time is significantly smaller. The cost of the total equipment configuration is comparable to that of typical large computer systems, but the cost per terminal is in the small computer range. In short, this system converts some of the raw power of the computer into condensed solution-time and greater creative power for the user.

OPERATIONAL DESCRIPTION

Equipment Configuration

The hardware (see Figure 1.1) consists of:

1. An IBM 7040[26] with 32K memory;
2. An IBM 1301[27] disk-file storage, for permanent retention of user programs;
3. An IBM 7320[28] drum storage, for the continual swapping of user programs;
4. A few magnetic tape units, for logging system transactions and to maintain normal computer capability;
5. An IBM 7740 communications control system,[29] for the real-time acceptance and transmission of messages;
6. A number of IBM 1050[30] terminals with keyboard-printer and, optionally, a card reader and card punch.

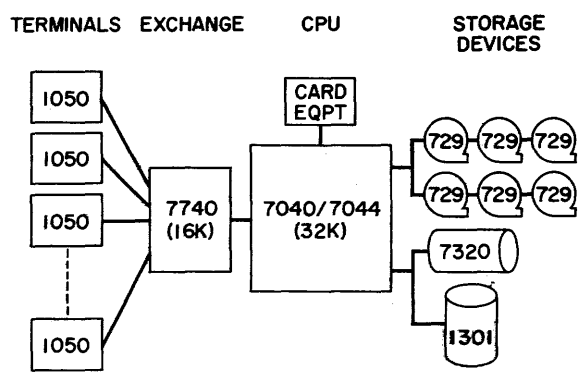


Figure 1.1. Remote-Computing System: Equipment Configuration

The User's Terminal Console

In use, the terminal console (see Figure 1.2) appears to be a self-sufficient FORTRAN machine. The user is completely unaware of any assembly system or the internal organization of the central computer. The language is consistent with FORTRAN, augmented by a set of operating, testing, and debugging statements. The mode of communication is called "conversational," as opposed to "batch," because the basic unit of input is the individual statement rather than an entire program, and every communication by one of the participants is acknowledged by the other.

The form in the terminal printer (see Figure 1.3) consists of 126 columns (10 characters/inch). The first 12 (unnumbered) columns are reserved for control fields, and the remaining 114 (numbered) columns are identical to a FORTRAN coding form, except for length.

The first five columns of the control-field portion are used to display a line number (101.0

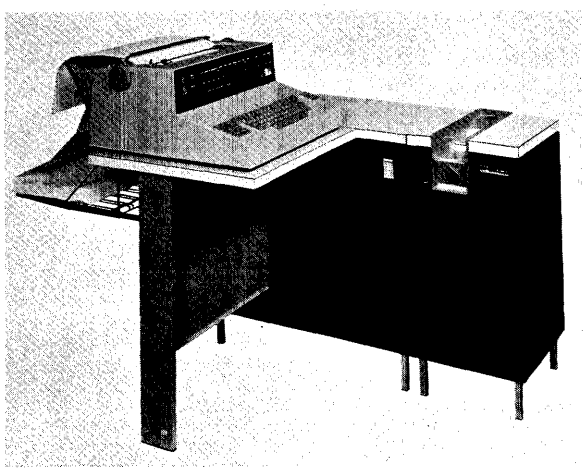


Figure 1.2. IBM 1050 Data Communications System (with card reader).

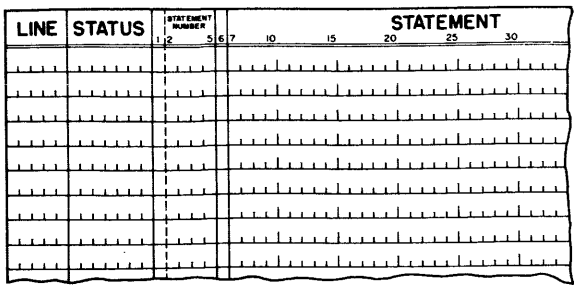


Figure 1.3. Remote-Computing-Terminal Programming Form.

to 999.9), i.e., an identifier automatically generated by the system to uniquely label each program statement. The first of the remaining seven control columns contains a code character —minus, plus, or equal—denoting the source of the line—input in Command or Program mode (see below), or output. The second through sixth columns of this field contain a status word which cues or informs the user: for example, the cue word READY invites the user to enter his next statement; and ERROR readily identifies a diagnostic message. (The seventh column is always blank.)

In the first two (numbered) columns of the FORTRAN-like portion of the form, "C blank" is treated as an ordinary comment, while "CV" causes the statement to be ignored by the system and serves as a hard-copy comment not germane to the program itself. Any character other than "C" in column 1 is considered a monitor-control character and the statement is treated as a normal comment; "CF" is treated as a comment in FORTRAN, but considered a normal statement in this system, and thereby serves to keep a source-program card deck compatible with other FORTRAN compilers.

General Operating Statements

The general operating statements (see Figure 1.4) may be used at any time. COMMAND establishes the Command mode (see below). EXIT signs the terminal off. Each terminal is set to standard real and integer formats (E15.8, I 11) for output of all values not under explicit FORMAT control. EDIT (---) permits the user to change either or both.

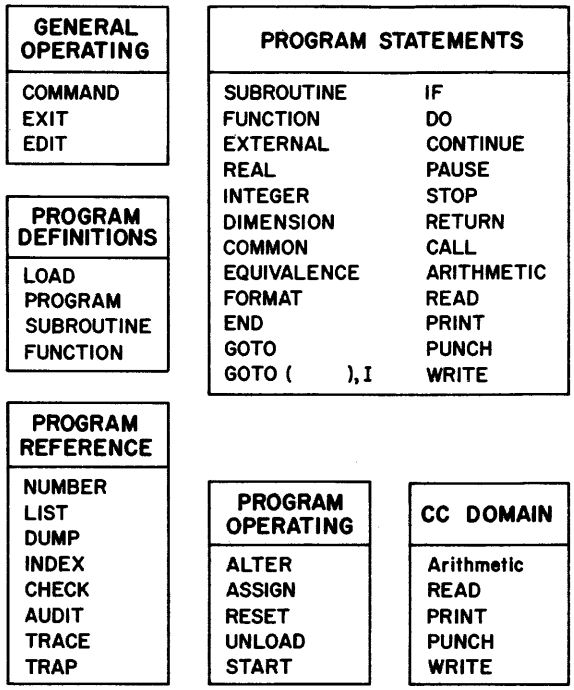


Figure 1.4. System Language.

Command Mode

When no program is active for a given terminal, that terminal is said to be in the Command mode; and, conversely, the entering of a COMMAND statement will destroy the active image of the user program. Since no program is active, statements cannot be retained, but must be processed immediately. Consequently, the user may employ only the general operating statements, the program-definition statements (see below), or a limited form of the arithmetic-assignment statement. This latter provision allows the terminal to be used as a fast, versatile symbolic desk calculator. In this mode, the user enters a statement of the form, X=any expression consisting of constants and built-in functions, and the system immediately evaluates the expression and prints the result on the user's terminal.

Program Mode

1. *Program-Definition Statements*
   The program-definition statements (see Figure 1.4) initiate the Program mode. LOAD fetches an existing program from the user's library, while PROGRAM initi-

ates the creation of a new main program, and SUBROUTINE or FUNCTION introduce subprograms.

2. *Program Statements*
   In the Program mode, certain statements, called program statements (see Figure 1.4), are translated and retained. Consequently, unlike the arithmetic assignment statement in the Command mode, their execution is initiated by the user. All FORTRAN statements used by this system (see Figure 1.4) are program statements. It is with these that the user may construct a stored-program solution to his problem; all other statements are processed immediately and are not retained. As with FORTRAN, there can be only one main program—but there can be numerous subprograms, with the restriction that no single subprogram exceed 4000 words of storage. In this way, although individual program size is restricted, total program size may be much larger.
3. *Program-Operating Statements*
   The program-operating statements (see Figure 1.4) allow the user to execute, alter, select I/O components, reset certain initial conditions, and unload his programs to library storage, at any time.

   START with various operands allows the user to begin execution from the first or any other executable statement, or to execute a segment from one line or statement number to another, or to resume execution after manual intervention.

   ALTER allows the deletion and insertion of statements.

   SELECT permits specifying the console's I/O devices other than the keyboard-printer.

   RESET with various operands initializes the program for fresh testing runs.

   UNLOAD places the user program in library storage, but does not remove it from the active status.

SOURCE LANGUAGE DEBUGGING

Debugging information is requested and displayed in a form consistent with the source programming language.[31, 32]

Diagnostic Structure

Errors committed by the user may be classified in two broad categories: syntactic and semantic.[33]

1. *Syntactic Errors*[34]
   All syntatic errors are considered the responsibility of the system and are further categorized as follows:

   Composition. Typographical errors, violations of specified forms, and misuse of variable names (e.g., incorrect punctuation, mixed-mode expressions, undeclared arrays, etc.)

   Consistency. Statements which are in themselves correct, but conflict with other statements (e.g., conflicting declaratives, illegal statement ending a DO range, failure to follow each transfer statement with a numbered statement, etc.)

   Completeness. Programs which have not been completely specified by the user (e.g., transfers to nonexistent statement numbers, improper DO nesting, illegal transfer into the range of a DO loop, etc.).

   Errors of composition and consistency are detected immediately upon entry of the offending statement. The user may immediately substitute a correct statement.

   Errors of completeness are discovered when the user signifies that his program is complete (by entering the END statement).

   Some errors (e.g., invalid subscript value, reference to an undefined variable, arithmetic spills, etc.) may, of course be detected only during execution. In this case, after a display of the error condition and its location, execution is interrupted and the terminal reverts to READY status. The user then has the option of either immediately correcting his error or proceeding with the rest of his program.

   For all syntactic errors, the diagnostic message is concise—in that the variable in error is named, or the column where the error occurred is specified—and often tutorial in suggesting the procedure for obtaining correct results.

2. *Semantic Errors*

   Semantic errors are concerned with the meaning or intent of the programmer and are necessarily his responsibility. However, an extensive set of debugging aids are provided for manipulating and referencing a program in ferreting out errors in logic and analysis.

Value Manipulation

Not too surprisingly, some types of program statements are also useful for manipulating the values of a user program (see Figure 1.4). Consequently, special characters—called process codes—may be inserted into the first two columns to allow the use of these statements as commands, thus causing values to be read into or out of selected variables. (This is analogous to the panel-entry/display functions performed at a conventional computer console.)

For example, "CC" in columns 1 and 2 of the FORTRAN-like form (Figure 1.3) has the following effect on its accompanying statement: the statement is immediately executed with all the effects of normal execution, but no new variable names are created; the statement is then discarded and does not become a part of the program. Thus, values may be inserted into factors or parameters at any time, thereby creating completely new testing situations without having to build their presence into the logic of the program or attempting to anticipate the debugging operations required.

Program Reference Statements

Program reference statements (see Figure 1.4) allow the user to display various vital conditions of his program. These statements are not retained. They are acted upon immediately, then discarded. Through their use, a complete, dynamic record of both control flow and data usage may be obtained.

DUMP (Figure 1.6b, line 143) produces an alphabetically ordered listing of program identifiers with their current values. Array subscripts are stepped automatically, contiguous zero-valued array elements are omitted, and empty elements (i.e., those never receiving a value) are flagged.

TRACE with various operands (Figure 1.6c, line 144 et seq.) allows the user during later execution to print out every change of value for all variables, for all variables within a specified region, or for specific variables whenever they are changed.

TRAP (Figure 1.6c, line 145 et seq.) is a logical trace of all control transfers, or of all transfers within a specified region.

NUMBER (Figure 1.6e) with various operands resequences and lists the program.

LIST generates a listing of the entire program or any specified portion of it.

INDEX (Figure 1.6f, line 231) produces a complete cross-reference table, ordered on statement numbers and variable names, showing the line number of every statement in which each statement number or variable name appears and whether it was declared, defined, or referred there; or, INDEX produces a single such line for a specified variable or statement number. Any line of the table which is, or may be, in error is marked with an asterisk. These features are very useful when making program modifications.

```
                COMMAND
101 -READY C    THIS IS A SAMPLE PROGRAM
101 -READY      PROGRAM SAMPLE
102 +READY      DIMENSION ZPLOT (51), TABLE(500)
103 +READY      DELX = .2
104 +READY      X = 0
105 +READY      Y = 1.
106 +READY  2 = 1
106 +ERROR 04200. STATEMENT NOT IN LANGUAGE.
107 +READY CV   TYPOGRAPHICAL ERRORS MAY BE CORRECTED IMMEDIATELY
108 +READY CV   BY SUBSTITUTING A CORRECT STATEMENT VIA KEYBOARD.
109 +READY      I = 1
110 +READY      READ 101, CHAR, ZPLOT
111 +READY  101  FORMAT(52A1)
112 +READY      PRINT 102
113 +READY  102  FORMAT(5X1HX5X1HY)
114 +READY      GOTO 2
115 +READY    1 PRINT 103, X , Y
116 +READY  103  FORMAT(2XF5.2,F8.5)
117 +READY CV    ANY STATEMENT OR SEQUENCE OF STATEMENTS MAY BE
118 +READY CV    VERIFIED BY IMMEDIATE EXECUTION AFTER ENTRY.
119 +READY      START 0
110 =I101  *
112 =0102       X     Y
114 =ERROR  TRANSFER POINT N DOES NOT EXIST
120 +READY      START 1
115 =0103     0.   1.00000
116 =CYCLE  END OF PROGRAM ENCOUNTERED DURING EXECUTION
121 +READY      I = I + 2
122 +READY      X = X + DELX
123 +READY      DELY = X*Y*DELX
124 +READY      Y = Y + DELY
125 +READY    2 TABLE(I) = X
126 +READY      TABLE(I+1) = Y
127 +READY      IF(X - 1.)1,1,3
128 +READY    3 DO 4 J = 1, I, 2
129 +READY      X = TABLE (J)
130 +READY      K=I+((TABLE(J+1)-TABLE(2))/(TABLE(I+1)-TABLE(2))*50.)
130 +ERROR  ARITHMETIC DECOMPOSITION ERROR(S)
130 +ERROR  MIXED MODE
131 +READY CV   STATEMENTS IN ERROR AT TIME OF ENTRY ARE NOT ACCEPTED.
132 +READY CV    SUBSTITUTION MAY BE MADE WITHOUT RE-ENTERING PROGRAM.
133 +READY      K=1.+((TABLE(J+1)-TABLE(2))/(TABLE(I+1)-TABLE(2))*50.)
134 +READY      ZPLOT(K) = CHAR
135 +READY      PRINT 104, X , ZPLOT
136 +READY  104  FORMAT(F5.2,5I4)
137 +READY    4 ZPLOT(K) = ZPLOT(K+1)
138 +READY      STOP 77
139 +READY      END
```

Figure 1.6a. Sample Program: Creation and Testing.

```
140 +READY        START 0
110 =I101  *
112 =Ø102        X      Y
115 =Ø103      0.    1.00000
115 =Ø103      0.20  1.04000
115 =Ø103      0.40  1.12320
115 =Ø103      0.60  1.25798
115 =Ø103      0.80  1.45926
115 =Ø103      1.00  1.75111
135 =Ø104    0.  *
135 =Ø104    0.20 *
135 =Ø104    0.40    *
135 =Ø104    0.60          *
135 =Ø104    0.80                *
135 =Ø104    1.00                          *
135 =Ø104    1.20                                      *
137 =ERRØR  VALUE ØF SUBSCRIPT IS ZERØ, NEGATIVE, ØR EXCEEDS DIMENSIØN
141 +READY        DUMP
    =             CHAR=-0.14191581E-08
    =             DELX= 0.20000000E-00
    =             DELY= 0.42026726E-00

142 +READY        EDIT(F8.5)
143 +READY        DUMP
     ERRØR    ILLEGAL CHARACTER IN TEXT
              DUMP
    =         CHAR=-0.00000
    =         DELX= 0.20000
    =         DELY= 0.42027
    =         I=          13
    =         J=          13
    =         K=          51
    =         X= 1.20000
    =         Y= 2.17138
    =         TABLE(1)= 0.
    =         TABLE(2)= 1.00000
    =         TABLE(3)= 0.20000
    =         TABLE(4)= 1.04000
    =         TABLE(5)= 0.40000
    =         TABLE(6)= 1.12320
    =         TABLE(7)= 0.60000
    =         TABLE(8)= 1.25798
    =         TABLE(9)= 0.80000
    =         TABLE(10)= 1.45926
    =         TABLE(11)= 1.00000
    =         TABLE(12)= 1.75111
    =         TABLE(13)= 1.20000
    =         TABLE(14)= 2.17138
    =         TABLE(500)= 0.
    =         ZPLØT(1)=-6.09524
    =         ZPLØT(2)=-6.09524
    =         ZPLØT(3)=-6.09524
    =         ZPLØT(4)=-6.09524
    =         ZPLØT(5)=-6.09524
 DUMP ALWAYS MAY BE INTERRUPTED.
```

Figure 1.6b. Sample Program: Creation and Testing (continued).

CHECK (Figure 1.6f, line 232) is an abbreviated INDEX in that only erroneous and suspicious items are displayed (i.e., only those INDEX lines marked with an asterisk).

AUDIT generates cross-reference information based on the execution of the program, showing which sections were never executed and which variables were never set, or set but never used. This concise, post-mortem summary of incomplete control flow and data usage is a powerful aid in ensuring the thoroughness of program debugging.

```
144 +READY        TRACE K
145 +READY        TRAP 101./138.
146 +READY        START 0
110 =I101  *
112 =Ø102        X      Y
114 =TRAP   TRANSFER TØ  2 (125 )
127 =TRAP   TRANSFER TØ  1 (115 )
115 =Ø103      0.    1.00000
127 =TRAP   TRANSFER TØ  1 (115 )
115 =Ø103      0.20  1.04000
127 =TRAP   TRANSFER TØ  1 (115 )
115 =Ø103      0.40  1.12320
127 =TRAP   TRANSFER TØ  1 (115 )
115 =Ø103      0.60  1.25798
127 =TRAP   TRANSFER TØ  1 (115 )
115 =Ø103      0.80  1.45926
127 =TRAP   TRANSFER TØ  1 (115 )
115 =Ø103      1.00  1.75111
127 =TRAP   TRANSFER TØ  3 (128 )
133 =TRACE        K=           1
135 =Ø104    0.  *
133 =TRACE        K=           2
135 =Ø104    0.20 *
133 =TRACE        K=           6
135 =Ø104    0.40    *
133 =TRACE        K=          12
135 =Ø104    0.60          *
133 =TRACE        K=          20
135 =Ø104    0.80                *
133 =TRACE        K=          33
135 =Ø104    1.00                          *
133 =TRACE        K=          51
135 =Ø104    1.20                                      *
137 =ERRØR  VALUE ØF SUBSCRIPT IS ZERØ, NEGATIVE, ØR EXCEEDS DIMENSIØN
```

Figure 1.6c. Sample Program: Creation and Testing (continued).

```
147 +READY        ALTER 137./137.
137 +ALTER        ZPLØT(K) = BLANK
1371+ALTER        ALTER 110.
1101+ALTER        BLANK = ZPLØT(1)
1102+ALTER        ALTER *
1102+ERRØR   DØ   128.0 REFERENCES UNDEFINED LABEL   4
148 +READY        ALTER 137./137.
137 +ALTER    4   ZPLØT(K) = BLANK
1371+ALTER         ALTER*
149 +READY        TRACE* K
150 +READY         TRAP* 101./138.
151 +READY        START 0
110 =I101  *
112 =Ø102        X      Y
115 =Ø103      0.    1.00000
115 =Ø103      0.20  1.04000
115 =Ø103      0.40  1.12320
115 =Ø103      0.60  1.25798
115 =Ø103      0.80  1.45926
115 =Ø103      1.00  1.75111
135 =Ø104    0.  *
135 =Ø104    0.20 *
135 =Ø104    0.40    *
135 =Ø104    0.60          *
135 =Ø104    0.80                *
135 =Ø104    1.00                          *
135 =Ø104    1.20                                      *
138 =S77
152 +READY
```

Figure 1.6d. Sample Program: Creation and Testing (continued).

```
                NUMBER 201.
201 =       CF    PRØGRAM SAMPLE
202 =             DIMENSIØN ZPLØT(51),TABLE(500)
203 =             DELX=.2
204 =             X=0
205 =             Y=1.
206 =             I=1
207 =             READ 101,CHAR,ZPLØT
208 =             BLANK=ZPLØT(1)
209 =           101 FØRMAT(52A1)
210 =             PRINT 102
211 =           102 FØRMAT(5X1HX5X1HY)
212 =             GØ TØ 2
213 =             1 PRINT 103,X,Y
214 =           103 FØRMAT(2XF5.2,F8.5)
215 =             I=I+2
216 =             X=X+DELX
217 =             DELY=X*Y*DELX
218 =             Y=Y+DELY
219 =             2 TABLE(I)=X
220 =             TABLE(I+1)=Y
221 =             IF(X-1.)1,1,3
222 =             3 DØ 4 J=1,I,2
223 =             X=TABLE(J)
224 =             K=1.+(TABLE(J+1)-TABLE(2))/(TABLE(I+1)-TABLE(2))*50.
225 =             ZPLØT(K)=CHAR
226 =             PRINT 104,X,ZPLØT
227 =           104 FØRMAT(F5.2,51A1)
228 =             4 ZPLØT(K)=BLANK
229 =             STØP 77
230 =             END
```

Figure 1.6e. Sample Program: Creation and Testing (continued).

```
231 +READY        INDEX
    =              1        +213.  -221.
    =              2        +219.  -212.
    =              3        +222.  -221.
    =              4        +228.  -222.
    =         *    5        +207.
    =            101        +209.  -207.
    =            102        +211.  -210.
    =            103        +214.  -213.
    =            104        +227.  -226.
    =          BLANK        +208.  -228.
    =          CHAR         +207.  -225.
    =          DELX         +203.  -216.  -217.
    =          DELY         +217.  -218.
    =          I            +206.  +215.  -215.  -219.  -220.
    =                       -222.  -224.
    =          J            +222.  -223.  -224.
    =          K            +224.  -225.  -228.
    =          *SAMPLE       201.
    =          TABLE         202.  +219.  +220.  -223.  -224.
    =          X            +204.  -213.  +216.  -216.  -217.
    =                       -219.  -221.  +223.  -226.
    =          Y            +205.  -213.  -217.  +218.  -218.
    =                       -220.
    =          ZPLØT         202.  +207.  -208.  +225.  -226.
    =                       +228.
232 +READY        CHECK
    =         *    5        +207.
    =         *SAMPLE        201.
233 +READY
```

Figure 1.6f. Sample Program: Creation and Testing (continued).

Built-in Subroutines

Since only program statements are retained, many excellent testing and debugging commands would not be available under program control, but would require the presence of the user at the moment of execution. To overcome this limitation, most of these statements have been designated as "built-in subroutines," a concept completely analogous to FORTRAN built-in functions. These statements, without change in their form, may be made the operand of a subroutine CALL statement. In this way, all the console testing and debugging features which may be of value are also available under program control.

COMPATIBILITY

Studious regard has been paid to maintaining consistency with other FORTRAN compilers. Programs written in the system language are acceptable without change to conventional FORTRAN IV processors. FORTRAN IV programs are acceptable to the experimental system with the following limitations:

1. The program must be written with statements from the system subset.
2. A restriction of all one-pass translators is that the source-deck ordering must have the declarative statements precede the imperative statements. Of course COMMENT and FORMAT statements may appear anywhere.
3. As in most compilers, the sequence of translated code for arithmetic expressions may differ from that produced by other compilers and slight discrepancies due to variations in truncations may occur.
4. Some minor differences in the internal representation of program constants, caused by different conversion routines, may also create slight differences in numerical results.
5. Individual source programs are limited to approximately 400 statements. This limit may often be circumvented by segmenting oversized programs into smaller subprograms.
6. Other Factors:
   a. No arithmetic function statements;
   b. No logical, complex, or double-precision variables;
   c. Number of elements (i.e., constants, variables, arrays, and functions) must be less than 190;
   d. Only one continuation card;
   e. No magnetic tape I/O;
   f. Some minor restrictions on equated variables;
   g. Constants—
      Reals: 8 digits, with magnitude within range 10^{-32} to 10^{32} or with zero magnitude,
      Integers: 10 digits;
   h. Array names must appear in a DIMENSION statement prior to any other appearance;
   i. Maximum I/O record size is 133 characters;
   j. Array names used as arguments must be declared in COMMON.

EXAMPLES

A program exhibiting many of the features available in this system is depicted in Figures 1.5 and 1.6. Figure 1.5a shows the final, correct version of the program. Figure 1.5b shows the correct output produced as a result of execution (see START statement, Figure 1.5a, line 128).

```
                COMMAND
101 -READY C    THIS IS A SAMPLE PROGRAM.
101 -READY C
101 -READY      PROGRAM SAMPLE
102 *READY      DIMENSION ZPLOT(52), TABLE(500)
103 *READY      X = 0
104 *READY      Y = 1.
105 *READY      I = 1
106 *READY      READ 101, DELX,CHAR,ZPLOT
107 *READY  101  FORMAT(F7.4,5A1)
108 *READY      PRINT 102
109 *READY  102  FORMAT(5X1HX7X1HY)
110 *READY    2 TABLE(I) = X
111 *READY      TABLE(I+1) = Y
112 *READY    1 PRINT 103, X, Y
113 *READY  103  FORMAT(2XF7.4,F8.5)
114 *READY      IF(X-1.)5,3,3
115 *READY    5 I = I + 2
116 *READY      X = X + DELX
117 *READY      DELY = X * Y * DELX
118 *READY      Y = Y + DELY
119 *READY      GOTO 2
120 *READY    3 DO 4 J = 1, I, 2
121 *READY      X = TABLE(J)
122 *READY      K=1.+((TABLE(J+1)-TABLE(2))/(TABLE(I+1)-TABLE(2))*50.)
123 *READY      ZPLOT(K) = CHAR
124 *READY      PRINT 101, X, ZPLOT
125 *READY    4 ZPLOT(K) =ZPLOT(K+1)
126 *READY      STOP 77
127 *READY      END
128 *READY      START 0
```

Figure 1.5a. Sample Program: Final Form.

```
106 =I101  00.0625*
108 =0102      X       Y
112 =0103      0.      1.00000
112 =0103      0.0625  1.00391
112 =0103      0.1250  1.01175
112 =0103      0.1875  1.02361
112 =0103      0.2500  1.03960
112 =0103      0.3125  1.05990
112 =0103      0.3750  1.08475
112 =0103      0.4375  1.11441
112 =0103      0.5000  1.14923
112 =0103      0.5625  1.18963
112 =0103      0.6250  1.23610
112 =0103      0.6875  1.28922
112 =0103      0.7500  1.34965
112 =0103      0.8125  1.41819
112 =0103      0.8750  1.49574
112 =0103      0.9375  1.58339
112 =0103      1.0000  1.68235
124 =0101  0.     *
124 =0101  0.0625*
124 =0101  0.1250*
124 =0101  0.1875 *
124 =0101  0.2500  *
124 =0101  0.3125   *
124 =0101  0.3750    *
124 =0101  0.4375     *
124 =0101  0.5000      *
124 =0101  0.5625       *
124 =0101  0.6250        *
124 =0101  0.6875         *
124 =0101  0.7500          *
124 =0101  0.8125           *
124 =0101  0.8750            *
124 =0101  0.9375             *
124 =0101  1.0000              *
126 =S77
129 *READY      UNLOAD
```

Figure 1.5b. Sample Program: Final Execution.

Figure 1.6 depicts a preliminary attempt to create and test this program. (All references that follow are to Figure 1.6.)

Input to the system may be from the keyboard or card reader at the remote terminals, or through input equipment located at the central computer. At line 106 a mispunched card causes printing of an error message. The user now suspends automatic input, substitutes a correct statement via keyboard, and then resumes automatic input. Of course, the substitution could have been made later by means of an ALTER (see below).

At lines 119 and 120, the user initiates intermediate execution and verifies his FORMAT statements before going further. In this manner, any statement, sequence of statements, DO loop, etc., may be debugged as the program is entered; or sections may be tested independently of the remainder of the program.

Execution of the entire program, line 140, discloses a number of bugs. Inspection of line 137 discloses the use of K as subscript. K could be printed selectively, but the user decides to dump all variables (see line 141). After DUMP starts, he interrupts it in order to change the format of the display and then dumps again (see line 143). In the event that the dump showing $K = 51$ is not a sufficient clue to the error, the user establishes a TRACE on K and a TRAP on the entire program, and starts again (see lines 144-146). This produces, together with his programmed output lines, a dynamic listing of control and data flow, before terminating with the same error message.

At line 147, the statements in error are corrected, but a statement number is inadvertently omitted. On terminating the ALTER status, a message is printed pointing out that the DO at line 128 references a nonexistent label. This error is corrected and a subsequent running of the program, line 151, shows that the subscript is now behaving properly.

There are other changes to be made, however. The NUMBER at line 152 yields a clean, renumbered listing of the current state of the program. Line 231 shows a complete INDEX, and line 232, the results of a CHECK statement. All of these will be helpful in reorganizing and documenting the final, correct version of the program.

Figure 1.7 shows the immediate evaluation of arithmetic expressions, consisting solely of constants and FORTRAN functions, in the Command mode.

EXPERIENCE

The system, from its most primitive form to the present, has been running for more than a year. A formal tryout of the system was run in November 1963 with 10 students attending IBM's Systems Research Institute.

```
                COMMAND
101 -READY      Y=2.5065*10.**(10.+1.)*EXPF(-10.)
101 =           Y= 0.11379485E 08
101 -READY      HENRY=2.E-9*50.*CLOGF(2.*50./10.)-1.0+10./50.
101 -ERROR 04117. PARENTHESES NOT IN BALANCE.
101 -READY      HENRY =2.E-9*50.*CLOGF(2.*50./10.)-1.+10./50.)
101 =           HENRY= 0.15025850E-06
101 -READY      ROOT1=(-25.+SQRTF(25.**2-4.*1.*2.))/(2.*1)
101 -ERROR ARITHMETIC DECOMPOSITION ERROR(S)
101 -ERROR MIXED MODE
101 -READY      ROOT1=(-25.+SQRTF(25.**2-4.*1.*2.))/(2.*1.)
101 =           ROOT1=-0.80257654E-01
101 -READY      HENRY=2.E-9*50.*CLOGF(2.*50./10.)-1.0+10./50.)
101 =           HENRY= 0.15025850E-06
101 -READY      VAL=1./COSF(50.)*LOGF(ABSF(SINF(50./2.)/COSF(50./2.)))
101 =           VAL=-0.97714996E 00
101 -READY      AREA=2.*10.*5.*SINF(3.1416/10.)
101 =           AREA= 0.30901768E 02
101 -READY      ARC=2.*SQRTF(4.**2+1.3333*2.**2)
101 =           ARC= 0.92375755E 01
101 -READY      ARC=2.*(4.*4.+4.*2.*2./3.)**0.5
101 =           ARC= 0.92376041E 01
101 -READY      S=-COSF(40.)**(20.+1.)/20./1.)
101 -ERROR 04117. PARENTHESES NOT IN BALANCE.
101 -READY      G=0.5*LOGF((1.+SINF(45.))/(1.-SINF(45.)))
101 =           G= 0.12594177E 01
101 -READY      S=SINF(45.)
101 =           S= 0.85090352E 00
101 -READY      G=0.5*LOGF((1.+.7071)/(1.-.7071))
101 =           G= 0.88135999E 00
101 -READY      E=20.*ATANF(20./4.)-4./2.*LOGF(4.**2+20.**2)
101 =           E= 0.15406644E 02
101 -READY      Q=(2./(3.1416*10.))**0.5*SINF(10.)
101 =           Q=-0.13726357E-00
101 -READY      Q=0.7978/SQRTF(10.)*SINF(10.)
101 =           Q=-0.13724918E-00
101 -READY C
101 -READY
```

Figure 1.7. Examples of Command Mode Operation.

USER	BACKGROUND SKILLS		SRI EXPERIMENT				
			7090 FORTRAN		REMOTE COMPUTING		
	FORTRAN	TYPING	NO. RUNS	NO. PROGRAMS DEBUGGED	NO. HOURS TRNG	NO. HOURS DBUG	NO. PROGRAMS DEBUGGED
A	HIGH	LOW	NONE	NONE	2	2	3
B	HIGH	LOW	5	3	2/3	1 1/3	2
C	HIGH	NONE	NONE	NONE	2	3	3
D	MEDIUM	HIGH	12	5	1	6	4
E	MEDIUM	HIGH	6	2	3	3	2
F	LOW	MEDIUM	NONE	NONE	2	6	5
G	LOW	NONE	10	3	2	4	2
H	LOW	NONE	4	2	2	1	2
I	LOW	NONE	3	1	1 1/2	F1/2	3
TOTALS			40	16	16	28	26

Figure 1.8. Results of SRI Tryout.

The students were divided into two groups, I and II, and given the same set of problems to be solved in FORTRAN. Group I was told to do the odd-numbered problems on the IBM 7090 and the even-numbered ones on the remote-computing terminals. Group II reversed this polarity.

The chart in Figure 1.8 shows the answers given by nine of the participants (the tenth failed to return his questionnaire) to the following questions:

1. "How much FORTRAN experience have you had?" (Answer was evaluated HI, LO, MED, NONE.)
2. "Have you had any typing experience?" (Answer was evaluated HI, LO, MED, NONE.)
3. "How many times did each problem go to the 7090 before you obtained correct results?" (Number of runs were summed.)
4. "How many problems did you debug on the 7090?"

5a. "Approximately how many hours of training did you have on the terminal console?"

5b. "How many debugging hours?"

6. "How many problems did you debug on the terminal console?"

Because this experiment was of limited scope, the experience reported must be taken cautiously. There are many variables which affect the usefulness and economy of this approach, and continuing field trials will yield more precise information.

SUMMARY

The time-shared use of a computer provides a convenient, economical service to numerous remote users. This access is enhanced by use of conversational, source-language debugging techniques. Although the experimental system is oriented to the IBM 1050 terminal, the FORTRAN language, and scientific applications, the techniques described are useful with other terminal devices, programming languages, and application areas. Preliminary operating experience indicates that systems such as the one described have considerable potential in enabling personnel less skilled in the programming art to rapidly obtain solutions to their problems.

ACKNOWLEDGEMENTS

Among the several people making significant contributions to the system, the authors wish to specifically acknowledge the work of Miss Geneva Butts, who implemented a major part of the source-language debugging features, and Mr. Murray Kizner, who implemented the translator routines for many Command features.

REFERENCES

1. E. G. ANDREWS, "Telephone Switching and the Early Bell Lab. Computers," *Bell System Technical Journal*, March 1963.

2. A RAND Symposium, "Economics of Remote Computing," *Datamation,* September 1961, October 1961, November 1961.
3. R. L. PATRICK, "So You Want To Go On Line," *Datamation,* October 1963.
4. D. B. BREEDON and P. A. ZAPHYR, "Pros and Cons of Remote Computing," *Control Engineering,* January 1963.
5. G. L. BALDWIN and N. E. SNOW, "Remote Operation of a Computer by a High Speed Data Link," *Proc. FJCC,* December 1962.
6. C. STRACHEY, "Time Sharing in Large, Fast Computers," *Proceedings of the International Conference on Information Processing—UNESCO,* June 1959.
7. J. MCCARTHY, "Time Sharing Computer Systems," *Management and the Computer of the Future,* Chapter 6, John Wiley & Sons, Inc., 1962.
8. F. J. CORBATO, "An Experimental Time Sharing System," *Proc. SJCC,* May 1962.
9. F. J. CORBATO, et al., "The Compatible Time Sharing System—A Programmer's Guide," The M.I.T. Press, May 1963.
10. W. V. CROWLEY, "Why Stretch?" *Proc. ACM National Conference,* September 1962.
11. S. BOILEN, E. FREDKIN, J. C. R. LICKLIDER, and J. MCCARTHY, "A Time Sharing Debugging System for a Small Computer," *Proc. SJCC,* May 1963.
12. W. CLARK and J. C. R. LICKLIDER, "On-Line Man-Computer Communication," *Proc. SJCC,* May 1962.
13. J. C. R. LICKLIDER, "Man-Computer Symbiosis," *IRE Transactions on Human Factors in Electronics,* March 1960.
14. L. C. CLAPP and R. Y. KAIN, "A Computer Aid for Symbolic Mathematics," *Proc. FJCC,* November 1963.
15. H. TEAGER and J. MCCARTHY, "Time-Shared Program Testing," *Proc. ACM National Meeting,* September 1959.
16. C. N. MOOERS, "The Reactive Typewriter," *ACM Communications,* January 1963.
17. G. J. CULLEN and R. W. HUFF, "Solution of Non-Linear Integral Equations Using On-Line Computer Control," *Proc. WJCC,* April 1962.
18. T. MARILL, D. EDWARDS, and W. FEURZEIG, "DATA-DIAL: Two-Way Communication with Computers from Ordinary Dial Telephones," *ACM Communications,* October 1963.
19. R. HEAD, "The Programming Gap in Real Time Systems," *Datamation,* February 1963.
20. W. A. HOSIER, "Pitfalls and Safeguards in Real Time Systems," *Datamation,* April 1962 and May 1962.
21. T. A. HALDIMAN, "Management Techniques for Real Time Computer Programming," *ACM Journal,* July 1962.
22. W. FRANK, W. GARDNER, and G. STOCK, Programming On-Line Systems," *Datamation,* May 1963 and June 1963.
23. D. ISRAEL, "Simulation Techniques for the Test and Evalution of Real Time Compiler Programs," *ACM Journal,* 1963.
24. R. HEAD, "Real Time Programming Specifications," *ACM Communications,* July 1963.
25. FORTRAN General Information Manual, IBM Form Number F28–8074.
26. IBM 7040/7044 General Information Manual, IBM Form Number D22–6645.
27. IBM 1301 Disk Storage, IBM Form Number D22–6576.
28. IBM 7320 Drum Storage, IBM Form Number G22–6717.
29. IBM 7740 Communications Control System, IBM Form Number A22–6753.
30. IBM 1050 Data Communications System, IBM Form Number A24–3020.
31. H. FERGUSON and E. BERNER, "Debugging Systems at the Source Language Level," *ACM Communications,* August 1963.
32. M. WILKERSON, "The JOVIAL Checker," *Proc. WJCC,* April 1961.
33. H. SCHWARZ, "An Introduction to ALGOL," *ACM Communications,* February 1963.
34. G. M. WEINBERG and G. L. GRESSETT, "An Experiment in Automatic Verification of Programs," *ACM Communications,* October 1963.

REMOTE COMPUTING—AN EXPERIMENTAL SYSTEM
Part 2: Internal Design

J. M. Keller, E. C. Strum, and G. H. Yang
Development Laboratory, Data Systems Division
IBM Corporation
New York, N. Y.

INTRODUCTION

This is the second of two papers dealing with the experimental remote-computing system. Part 1 described the system as viewed by a user who is unaware that he is jointly sharing the central computer with numerous other users. This paper (Part 2) describes the internal design of the system, with attention focused on those features which are of general interest and applicable to the design of other programming systems.

This paper is introduced by a description of the over-all control structure and data organization. Each of the principal subsystems is then described. The paper concludes with some remarks regarding possible extended applications. An appendix describes in some detail the algorithms used in the decomposition/recomposition of arithmetic expressions.

OBJECTIVES

An operating system servicing numerous online users must meet certain design objectives that might be regarded as secondary or even unnecessary in conventional operating systems or compilers. But these objectives become paramount when the psychological and practical effects of sustained, immediate access to a computer are considered. Thus primary attention must be given to attaining:

1. Immediate error diagnostics;
2. Program alteration without recompiling;
3. Extensive symbolic debugging aids;
4. Ready availability of the source version of the user program;
5. A user program that is:
   a. dynamically relocatable,
   b. easily interruptible, and
   c. storage protected.

SYSTEM ORGANIZATION

Programs

The experimental remote-computing program is divided into three major system areas (Figure 2.1):

1. The *Scheduler,* which is responsible for maintaining awareness of the total system status and for ordering and assigning tasks to the other system parts;
2. The *Process Control system,* consisting of the Translator, which reduces the user's input statements to an equivalent internal form (see below); the Interpreter, which executes this internal form; and the Process Control program, which regulates these two subsystems on the local level;
3. The *I/O Control system,* which is responsible for monitoring and operating all

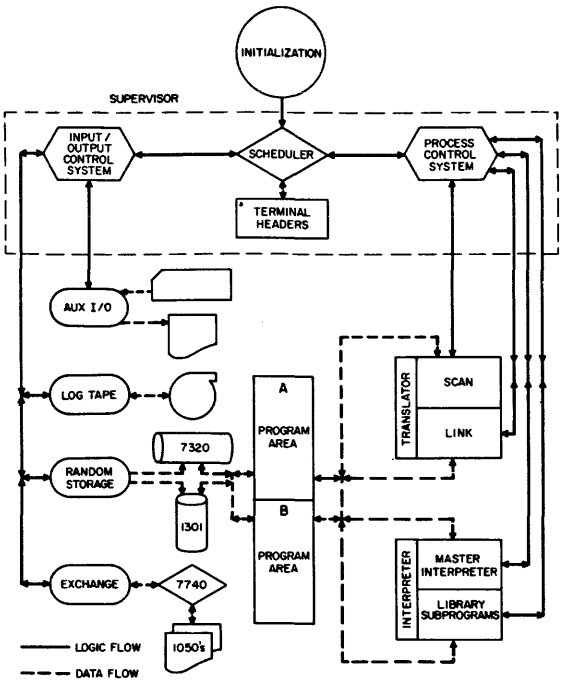


Figure 2.1. General Block Diagram of System.

I/O attachments, including the communications exchange.

Data Organization

At the system level, there are three principal data constructs:

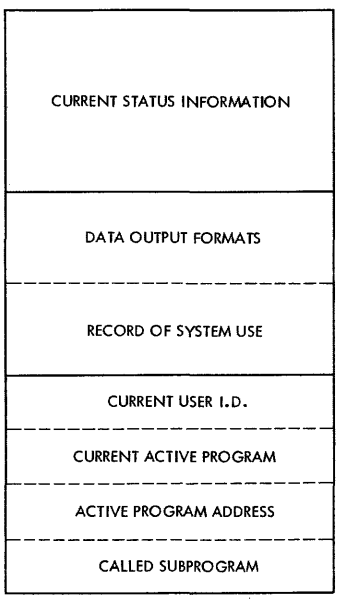


Figure 2.2. Terminal Header.

The Terminal Header

For each terminal in the system, there is a Terminal Header record (Figure 2.2) containing the following information:

1. Current status:
   a. operating mode, i.e., Command or Program (Rf. Part 1),
   b. terminal status, i.e., I/O wait, busy, or dormant,
   c. control information, i.e., should the system interrupt automatic status (execution) and return to manual status (statement entry) or continue automatic status,
   d. type of terminal component active,
   e. terminal ID,
   f. storage allocation block;
2. Header information for Command mode execution:
   a. formats for data output,
   b. system use records;
3. Temporary locations for random storage access:
   a. current user identification,
   b. name of current active program,
   c. location of active program for this terminal,
   d. name or location of subprogram called by current program.

The Master Block

For each statement in the language (Rf. Part 1), there is a Master Block record (Figure 2.3) containing the following information:

1. A statement type identifier;
2. A statement class identifier;
3. The symbolic, external statement identifier with associated control characters for recognizing the statement name on input and for recreating it on output;
4. Various indicators which denote intrinsic statement characteristics for checking purposes;
5. Addresses for transfers of control to the various major system routines, e.g., Translator, Interpreter, etc.

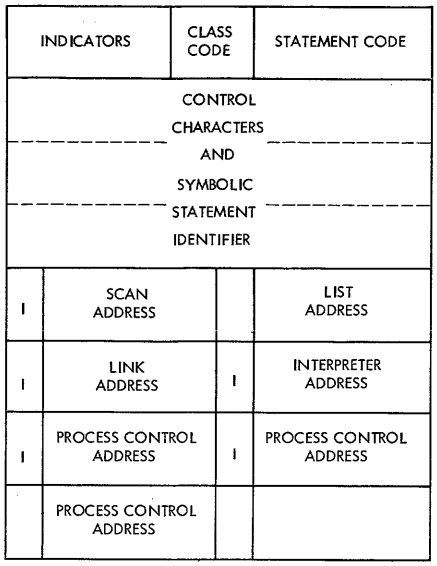


Figure 2.3. Master Block Record.

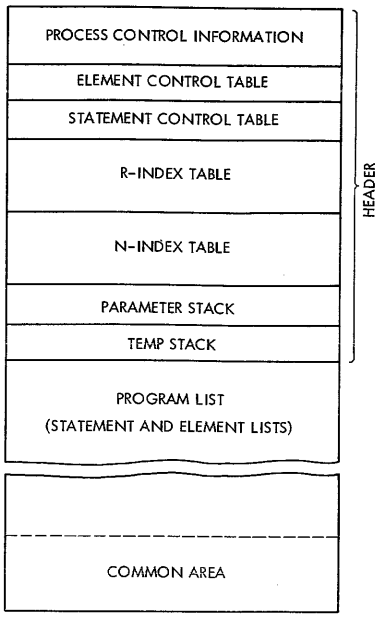


Figure 2.4. User-Program Layout.

The Master Block is used either as a dictionary, when information concerning the statement is needed, or as a switching center, when control flow within the system is dependent upon the statement type. A single record for both of these activities provides considerable flexibility in adding new statements, in modifying control conditions, and in making basic system modifications.

The User Program Layout

For the entire system there are two large, fixed areas (Figure 2.4) reserved for occupation of the various active user programs, Programs brought into these areas are relocated under program control; all I/O to and from these areas is overlapped. The duration of occupancy is determined either by overstepping a time limit or by the occurrence of one of several specific conditions (see following section on user-program organization).

The layout of the user program is divided into two parts:

1. The statement and element records (see following section) which comprise the user program; and
2. The header.

The header is further subdivided into two parts:

1. List control and other controls for the program records (see following section); and
2. Control words used by the Process Control program to keep track of program status.

USER-PROGRAM ORGANIZATION

The User Program

The user's source-program statements are mapped into equivalent internal records, which are classified and controlled by list structures. These records and their controlling elements constitute the user's program (see Figure 2.4). Every statement of the user's program is reduced to an individual statement record; and every element (name or label) is reduced to an individual element record. These records are inserted and chained on lists in the program area in the order of their appearance and creation. Control is maintained through tables of list-control words in the header portion of the user program. All addresses in the user program are relative to its base in order to facilitate relocation.

Records

Element Records

An element in the source program is defined as a label, constant, variable, array, or function name. Every source element maps into a fixed-length, internal element record (see Figure 2.5) containing the following information:

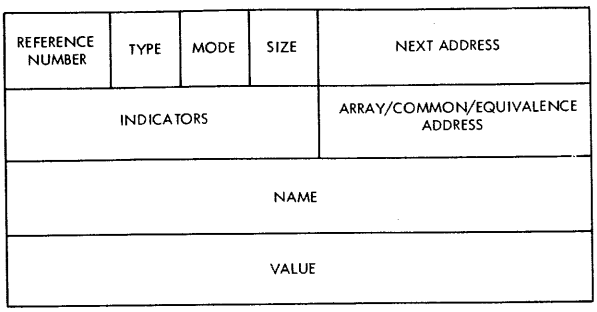
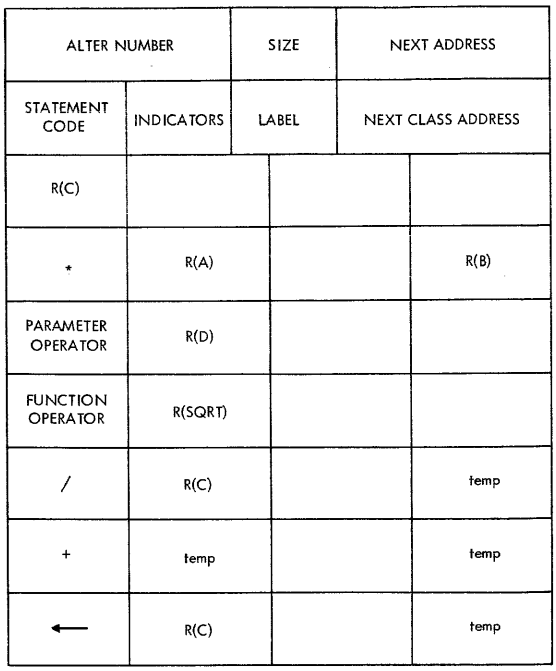


Figure 2.5. Element Record.

1. Reference number—a unique internal numeric identifier, assigned by the system. It is used for all internal referencing by the system.
2. Type—denotes the type of element, i.e., label, constant, variable, array, function.
3. Mode—denotes the mode, real or integer, of elements referring to numeric quantities.
4. Indicators—contains properties attributed to the element by declarative statements and/or execution. These include storage-allocation, and indications of element usage at object time.
5. Name—the external alphanumeric identifier.
6. Value—either the numeric value of the element or supplemental information for an array or function.
7. Next address—address of the next element record.
8. Array COMMON EQUIVALENCE—address of the value, if the element is in COMMON or is an array; or an offset address, if the element is equated to an array.

Statement Records

Every source statement maps into a variable-length, internal statement record (see Figure 2.6), which contains in coded form all information present in the source statement. Each record begins with two standard words containing the following information:



C = A*B + C/SQRT(D)

Figure 2.6. Statement Record.

1. Alter number—a unique internal numeric identifier assigned by the system. It denotes the position of the statement relative to all others in the program; it is referenced by the user when modifying the program, manually requesting information, or starting execution.
2. Statement code—identifier of the particular statement type.
3. Indicators—reflects usage of the statement during execution.
4. Label—refers to the associated external statement number, if any.
5. Next address—address of the next statement record.
6. Next class address—address of the next statement record of the same type.

The remainder of each statement record contains one or more words. Their number and content depend on the particular statement type. For example, an arithmetic-statement record contains the macro representation of

the translated expression, while a DO statement contains references to the indexing parameters.

Lists

The objective of providing for alteration of individual statements was the deciding factor in determining the internal record organization. The conventional table-oriented approach appeared much less attractive than the classification of records by lists.[1, 2, 3, 4, 5, 6]

Most compilers use tables to record information necessary for referencing and validating data usage and control flow. In this system, the same information is kept in the statement and element records. However, organizing these records on lists allows for increased flexibility in the compiling system.[7, 8, 9, 10] For example, deletion and insertion of statements for program modification is easily provided. Time-consuming recompilations become completely unnecessary as a result of this altering provision. In addition, errors resulting from improper control flow and from invalid variable references can be diagnosed earlier in the compilation process than is common with conventional compilers.

Element Lists

Each element record is chained onto one of 26 element lists, each list consisting of all those element records whose symbolic names have the same initial letter. Element records within each list are ordered alphabetically by symbolic name. There are two additional lists which link numeric elements as either integer or real constants. This set of element lists provides two significant features:

1. The symbol look-up is more efficient since only the set of symbols with the same initial letter are considered;
2. Fully alphabetized symbolic cross-reference listings and memory dumps are easily provided.

Statement Lists

Each statement record is chained onto two lists:

1. The entry list consisting of all statement records in source sequence (i.e., ordered by "alter number");
2. One of the class lists, consisting of all statements of a particular class (e.g., arithmetic, control, DO, I/O, allocation declarations, etc.).

The entry list is used both to control execution sequence and to provide the proper ordering when the source program is reconstructed from the internal form.

The class lists are extremely useful in performing checking operations (e.g., checking DO loops for proper nesting and control transfers).

List Control

Every list is controlled by a single control word pointing to the first and last records. For the statement lists there is a small table of control words for statement control (see Figure 2.4). Another similar table controls the element lists. In addition there is also a master table controlling the symbolic names of reserved system symbols: library functions (e.g., SIN, SORT, etc.); built-in functions (ABS, FLOAT, etc.); and system subroutines (DUMP, EXIT, etc.).

Addressing

Two tables exist for control of the element and label identifiers (see Figure 2.4). These are the R-index, or internal-identifier reference table, and the N-index, or numeric-label table. For every element that appears in a program, an entry for its internal identifier, R, is made in the R-index table; similarly, for every statement label, N, an entry is made in the N-index table.

Every element or statement in the program can be accessed in an "asociative" manner by sequentially searching the lists until a match is found for the requested symbolic name or alter number. Each element or labeled statement can also be located in a "direct-look-at" manner[11] by using the internal identifier for the element or label as an entry to the R- or N-index table. Thus the flexibility of associative list searching and the efficiency of direct element fetching are both incorporated in the system.

FUNCTIONAL DESCRIPTION

The Scheduler

The purpose of any real-time, multiprogramming supervisory program is to synchronize, control, and monitor system operation.[12, 13, 14] The program is responsible for determining what things are to be done, and by whom, to what, where, and when each is to be done. It has the duty of maximizing system throughput and ensuring reliable operation, and, in this case, of maintaining rapid and level response times at the terminal consoles.

At the nucleus of this supervisory structure (see Figure 2.1) is the Scheduler,[15,16] which controls the:

1. Process Control program, which in turn directs the processor routines that translate and execute user programs; and
2. I/O Control program, which coordinates the communications exchange, random storage devices, tape units, reader, and online printer.

The Scheduler performs continual sequential sampling of the subsidiary subsystems and maintains pertinent status data in the terminal headers. When data has been received from the terminal, the Scheduler examines the terminal header and decides whether to transmit a request to the random-storage I/O queue to fetch the user program (Program mode), or, if no program is required, save the data in a to-be-processed queue (Command mode).

In either event, the Scheduler passes to the Process Control program all information necessary for processing the input message—such as locations of the terminal and program headers, the location of the input message, and the operating mode of the terminal.

Even if no message has been received for a given terminal, its active program will be fetched from random storage and the Process Control program entered, if the terminal header shows that the program is in the automatic state (i.e., in the process of execution). After each return from the Process Control program under this condition, the Scheduler must determine whether the automatic state should be terminated.

There are two kinds of termination: temporary, and return-to-manual. Temporary interruption frees the system for use by another terminal and may occur for the following reasons:

1. The allotted time interval has expired;
2. Input data is requested:
3. Output buffer is filled;
4. An external subprogram is invoked.

The return to manual status occurs when:

1. An error condition occurs;
2. A STOP or PAUSE statement is executed;
3. The end of the program is encountered;
4. The user requests an interrupt from the terminal.

The Process Control Program

The Process Control program (see Figure 2.7a) accepts information from the Scheduler and coordinates the activities of the processor programs. All of the appropriate Process Control routines and service routines must be initialized (1) to process the terminal header if the termnial is in the Command mode, or (2) to process various parts of the program header and list if in the Program mode. The Process Control program maintains an action code in

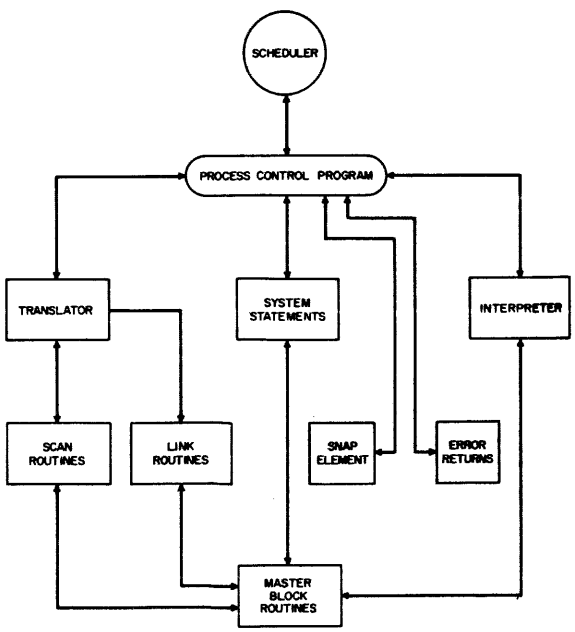


Figure 2.7a. General Diagram of Process Control Flow

each header to determine the next task to be performed upon the user program. The Process Control program may require the input statement just received to be scanned by the translator; it may require continuation of execution, of a DUMP, or of a LIST. The user program may be in the ALTER mode; it may be in the process of being tested for certain conditions which may prohibit further execution. Data for an input statement may be awaited or output of multiple-error messages may be in effect.

When the user program is in the manual mode, the Process Control program has the responsibility of examining the process codes returned from the translator and of taking the necessary action. When the user program is in the automatic mode, execution may be temporarily halted and, in some cases, the program may be returned to manual-mode status (see above). When a subprogram "call" is made, execution halts until the next cycle for this terminal. At that time, the called subprogram becomes the user's active program and is brought into memory in place of the calling program. When a RETURN is effected or if an error occurs, the calling program is reactivated.

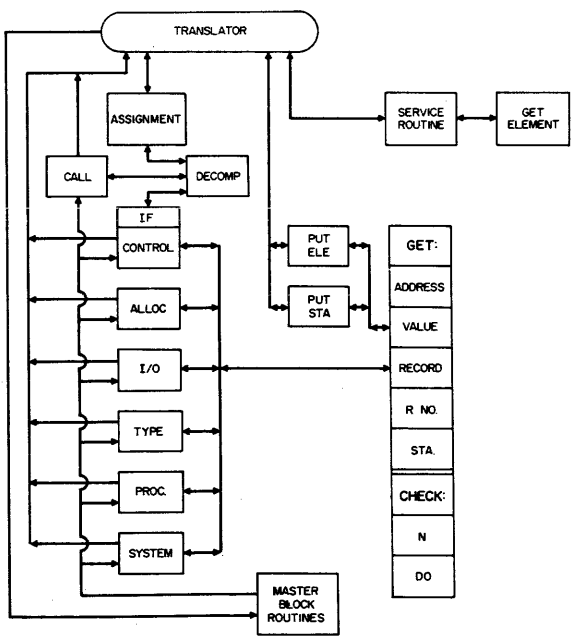


Figure 2.7b. General Diagram of Process Control Flow —Translator.

In order that the user may always be aware of the status of his program, condition codes are printed at the terminal whenever a change of status occurs. He is notified when input (a statement or data) is requested; when an error occurs; when and why execution was terminated; when a system statement (see Figure 2.7d) occurs (e.g., DUMP, INDEX, TRACE); and, optionally, when a subprogram call is made. When execution runs off the end of a program, and when a STOP or PAUSE is encountered, he is informed that his request for interruption of execution has been recognized. In short, the Process Control system always knows what is happening in the user program, and continually keeps the user informed of the status of his job. The objective is to provide the remote user with a more complete awareness of his program's status than is obtainable at a conventional computer console.

The Translator

Scan Routines

The Translator (see Figure 2.7b) is responsible for transforming the source-language program to the internal form.[17, 18, 19] (See Figures 2.5 and 2.6.) A preliminary scan is first used to identify arithmetic statements. For all other statements, the statement operator is collected and used to reference (via a Master Block routine) the corresponding master record. Control then passes to the translation routine for the particular statement type, e.g., GOTO, RETURN, PRINT, DIMENSION, etc.,

Every statement's decomposition goes through the same basic phases to form element and statement records. These involve the use of several service routines to collect the element name, find its record in a list or create a new record, and validate the statement and element usage.

As each element in a statement is collected, a search is made to determine if it has previously appeared in the program. If the element has been previously used in the same statement, the record appears on a current element working list; otherwise, it may be found on the element list in the user's program or, alternately, on the master list of reserved and

system names. If no record is found, then the element is new to the program, and a record is created for it and placed on the current working list. The information put into the record depends on the variable itself and the kind of statement it appears in. The mode indicators for an element depend either on its initial letter or on its appearance in an INTEGER or REAL declarative statement. The type and variable indicators depend on the statement type. A variable appearing in a storage-allocation statement is nagged according to its declaration as an array, common, or equated variable.

If all source elements have previously appeared, no new element records result from the translation of a statement. However, a statement record must always be created. The Master Block record for the statement type provides the statement code. Statements that involve a list of variable, such as DIMENSION, EQUIVALENCE, or COMMON, contain a count of the variables used followed by the R-number of the variables.

Statements which involve a list of labels, such as:

GO TO (5, 6, 7, 8), I or IF (J–5) 12, 3, 12

contain an item count followed by the numeric labels. If the execution of a statement will change the value of a variable, the identifier of that variable is placed in a special field in the statement (see Figure 2.6).

Statements containing arithmetic expressions or input/output lists involve specialized decomposition routines. The master Translator passes to both these routines essentially the same input: a string of words, each word containing either an operator or an R-number. The decomposition routine transforms these elements into an ordered set of arithmetic macros consisting of an operator and two operands in every word. These macros are then returned to the master Translator and added to the statement record (see Figure 2.6). The input/output list-decomposition routine also returns a macro set of executable operations, (A detailed description of the arithmetic decomposition is contained in Appendix I.)

Scan Diagnostics

Throughout the translation-scan phase, checking occurs for syntax and composition-type errors. Illegal statement operators and invalid statement forms are detected early in the translation. Lack of a label on a FORMAT statement or the presence of a label on a declarative (where control may not flow) violate the definition of the statement type. Illegal uses of variables, such as a simple variable name followed by a parenthesis, are detected by testing indicator bits in the element records. The same checking of element records is used to detect mixed-mode errors in the arithmetic expressions. The number of subscripts following an array name is checked for agreement with the number declared in the DIMENSION statement for that variable. In general, the Translator detects all syntactic errors which are within the context of a single statement and those semantic errors which occur in the use of the elements in the statement.

Link Routines

If the statement has no errors, the Process Control program decides whether to save the statement record and its related element records as part of the user's program. A statement record is either added to the end of the entry and class lists or, if the ALTER mode is active, is inserted somewhere into these lists. To accomplish this linking, space for the new record is found, and the address of this area, relative to the program area base, is inserted as the "next" address in the preceding record on the list.

If the statement record is successfully put into the program area, the element records are linked to their respective lists. Every new element record also causes its address (relative to the user program base) to be entered into the R-index table.

Link Diagnostics

Before a new record is actually chained to a list, certain checks for consistency of referencing are made. These are partially accomplished through use of the N-index table, in which all references to labels are recorded. For every label in the program there is a corresponding

entry in this table. In each entry there are two fields: the first specifies the relative address of the labeled statement record; the second specifies how the label is referenced, i.e., from an I/O statement, a DO statement, or a branch type statement. These entries are set up and checked before the statement is linked to the lists.

Examples of the errors detected at this phase are:

1. Duplication of statement numbers;
2. Referring to a FORMAT statement from a branch statement;
3. Referring to an executable statement from an I/O statement;
4. Using an illegal statement as the end of a DO (e.g., a branch type);
5. Referring, as the end of a DO, to a statement which precedes the DO.

Another type of consistency error detected at this time is based on ordering of statements. To link a statement into a list, the preceding statement must be available. In the case of an ALTER insertion, the succeeding statement is also available. It is possible, then, to check for violation of such precedence rules as:

1. Declarative statements must precede executable ones;
2. The first executable statement following a branch-type statement must be numbered (i.e., every section of the program should be potentially executable).

It is important to note that all these consistency and precedence errors are reported to the user immediately after the statement is accepted by the system. Most conventional compilers report all composition-type errors throughout the entire program before going on to check for consistency errors. In this system, diagnostics are provided as early as possible.

When the END statement is first linked to the program list, or thereafter at the end of an ALTER sequence, several specialized routines check completeness of control flow and data referencing.

Storage Assignment

The value of a simple variable or a constant is stored in the element record. However, storage for all arrays and any variable appearing in common must be specially assigned. Because of their interaction, all allocation declarations must be entered before storage can be assigned; on the other hand, storage must be assigned as soon as possible since partial execution of the program may be requested at any time. The Link routine, on recognizing the first executable statement, assigns storage on the basis of all declarative statements, which are linked on the same class list. After an ALTER sequence involving a storage allocation the same operation is again performed.

Storage Control

When a statement or element record is to be linked to a list, it is moved from a temporary working area to the program area. Space for successive records or data storage is at first assigned sequentially throughout the program area.

When the user deletes (via ALTER) any statement or variable from the program, the associated records are unlinked from the program and chained to a "null" list ordered by size of record. When space is needed for a new record, the null record that best fits (i.e., large enough but with minimal "trim") is selected; this technique prevents wasteful fragmentation of the null-storage areas.

If no record on the null list satisfies the space requirement, but the total size of the scattered null records would provide enough space, then a "squeeze" is performed by moving every record in the program to a contiguous storage area. All references to relative addresses in the program area are then changed to reflect this relocation.

The Interpreter

Execution of the user's program is done in an interpretive fashion[20,21,22,23,24] on a statement-by-statement basis under control of the Process Control program. This control program sends to the Interpreter the address of the statement to be executed. Upon successful

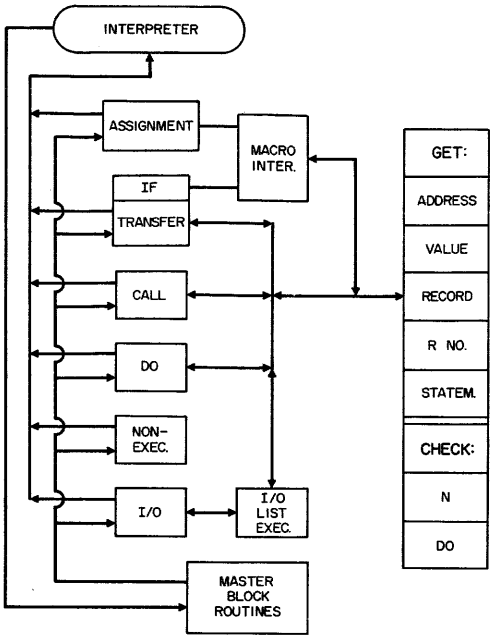


Figure 2.7c. General Diagram of Process Control Flow —Interpreter.

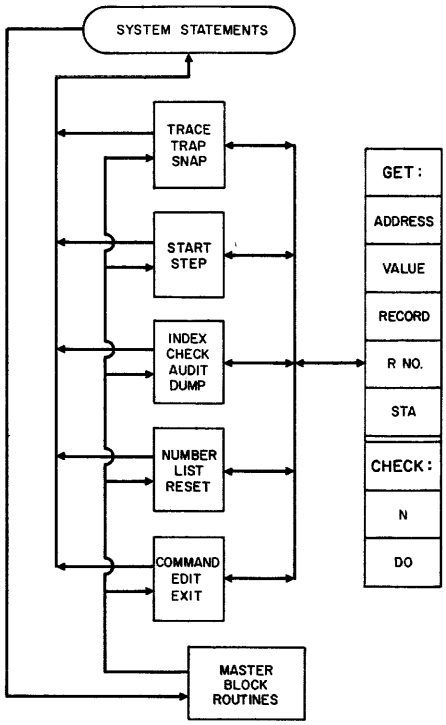


Figure 2.7d. General Diagram of Process Control Flow —System Statements.

execution of this statement, the relative address of the next statement is saved in the program header. At this time, an indicator is turned on in the statement record, showing that the statement has been executed.

The Interpreter can be broken into several parts (see Figure 2.7c):

1. The master interpreter, which decodes the statement type;
2. The service subroutines used by all statement routines;
3. The macro interpreter used for arithmetic expressions;
4. The various statement routines.

Decoder

To interpret a statement, a code is fetched from the statement record and, using the Master Block, control is transferred to the appropriate Interpreter routine.

Service Routines

These subroutines are used to fetch element and statement records and to address value words for variables and constants. In the Interpreter, all fetching is done by a direct "look-at" of a table entry for an address. This is in contrast to the associative referencing used in the Translator. Execution speed is considerably increased by this elimination of list searching.

All references to a variable in a statement record are by its internal identifier. This number is used as a key to the R-index table to access the relative address of the element record. Whenever a value word is fetched for use, an indicator for "variable used" is turned on in the element record. Similarly, if a value is stored into a value word, an indicator for "variable set" is turned on.

Macro Interpreter

The evaluation of an arithmetic expression can be expressed in hardware terms; i.e., the system has an instruction repertoire of six two-address instructions, and is equipped with a group of pushdown registers.[25, 26, 27, 28, 29] Execution of the statement in the interpretive

mode is analogous to machine execution and involves several steps:

1. Fetch the next macro (i.e., instruction) to be executed;
2. Fetch operand values;
3. Decode the instruction operation;
4. Perform the specified operation;
5. Store the result in a push-down stack.

Subscripts of arrays and arguments of functions are indicated by a special operator. When this operation is encountered, an entry is made in a push-down parameter stack. Values are fetched from this stack for computing an array address or for passing arguments to a function.

Functions are also indicated by a special operator. When this operator is encountered, the function-element record is fetched. All records for library routines point to their actual machine coding within the system. All other functions are called from random storage.

Statement Routines

Every statement has a particular Interpreter routine associated with it. There are several categories which should be discussed:

1. Arithmetic—the macro interpreter is used to evaluate the expression to the right of the "=" and store its value in the left-hand variable.
2. Branch—the macro interpreter is used to evaluate the arithmetic expression for an IF; a service routine is used to fetch the value for I in GO TO (....), I. The proper transfer point is chosen from the list of numeric labels in the statement. This numeric label is used to access the N-index table for the relative address of the statement to which control should flow.
3. DO loops—the initial execution of a DO statement creates an entry in a push-down stack controlling DO nesting; it also initializes the value of the DO index variable and flags its element record as an active DO index. The execution of the last statement in the range of a DO is detected through checking of an indicator turned on in the translation process. After execution of this last statement, the DO statement is fetched again and its index is tested and incremented. Execution continues with the statement following the DO until the indexing condition is satisfied.
4. Input/Output—an I/O macro interpreter is used to compute addresses of values to be passed to the appropriate input/output service routine. A table is generated in the execution process to handle variables in the list controlled by implied DO's.

Execution Diagnostics

Choosing the interpretive approach to execution necessarily means sacrificing speed. For debugging purposes, this is not often a serious impediment—especially since diagnostics are possible for many errors never detected in conventional execution. These include detecting:

1. A value word not being set before used:
2. A subscript value not being valid;
3. A DO index being reset in the range of the DO;
4. A computed GO TO parameter not being in range;
5. The size of an integer exceeding its limits;
6. The existence of an illegal value in an I/O list with implied DO's.

Input-Output Control System (IOCS)

The prime responsibility of the Input-Output Control system is to select, from the respective queues built up by the Scheduler, the next task or combination of tasks to be performed by the individual I/O units. Upon completion of a given task, the Scheduler is notified either directly through program switch indications or indirectly through the terminal header. Before relinquishing control to the Scheduler, the IOCS initiates the next task for that channel device based on the queue information. It also maintains control surveillance over all I/O buffer areas to prevent overflow.

The I/O attachments consist of disk, drum, magnetic tape, card reader, on-line printer and communications exchange.

The disk is used as a permanent storage for user programs. The drum serves as a rapid

access storage device for the repeated shuffling of user programs in and out of memory. The magnetic tapes, card reader, and printer are used in a conventional manner.

The communications exchange has some interesting capabilities not found in more conventional I/O equipment.

The Exchange

The IBM 7740 communications control system[30,31] is used to buffer and control the traffic flow between the communications network and the IBM 7040 computer.

It is a stored-program computer with a rather specialized instruction repertoire designed for real-time applications. The instructions possess powerful logic and data manipulating facilities, through somewhat limited arithmetic capability. Instructions are fixed in size, one instruction per 32-bit word, while data is composed of strings of 8-bit characters. Addressing is at the character level, up to a maximum of 64K characters (i.e., 16K words).

The 7740 program performs several communications-oriented functions. First, it accomplishes line and terminal control by generation, recognition, and manipulation of control characters, in order to establish a connection to the remote terminals, and to determine the operation to be performed. Second, it provides message control, so that the messages may reach their intended destinations: they are logged in, monitored for correctness, and converted from the various transmission codes to the codes acceptable to the other devices in use. Third, it provides protection to ensure the proper disposition of messages, and to ensure the correction of transmission errors wherever possible.

To simplify these functions, the 7740 has several hardware and programming capabilities not often found in conventional computers.

1. The most striking of these is the ability to operate in an independently controlled hierarchy of modes. In increasing order of priority (that is, decreasing order of interruptability), these are:
   a. The normal mode. The normal activities involved in polling, addressing, and monitoring of all communications devices are conducted in this mode. Because of the large number of lines, processing is on a continuous service basis, whereas a conventional computer attains I/O overlap by yielding independent control to the devices and servicing them on an interrupt basis.
   b. The I/O mode. This mode is used to control input/output between the 7740 and the 7040. A special uninterruptable state, called copy mode, is used for the actual transmission of information.
   c. The attention mode. This mode is entered when service (not connected with any hardware malfunction) is needed (e.g., servicing the interval timer).
   d. The service mode. This mode is entered if malfunctions are detected.

   Entry to the service, attention, or I/O copy modes may be initiated by the machine; entry to any mode may also be initiated by the program. In addition, it is possible to inhibit mode change so that tables or programs used in several modes may be protected (this is analogous to disabling a channel on a conventional computer).

   Associated with each mode is a pair of machine registers which contain the complete status information. Mode change is automatically accomplished by storing this information into the cells associated with the old mode and picking up the corresponding information from the cells associated with the new mode.
2. Time-stamping, essential to control in any communications or real-time environment, is provided for by the interval timer, which is automatically updated by the machine every few milliseconds. This timer is used in conjunction with attention-mode programs to provide a programmed real-time clock, and a programmer-accessible interval timer.
3. Information about each of the communication channels (or lines) is maintained in fixed positions of core storage using two channel-control words, one pair for each line involved. The current status informa-

tion of these words is manipulated by both the hardware and the programs in order to control the flow of information within the system.

4. In order to facilitate the acquisition and transmission of data, the memory of the 7740 is considered by the hardware to be divided into blocks of 32 characters, each of which begins on an 8-word boundary. The first 30 characters of each block are used to store data, while the last two provide a 16-bit chaining address used to indicate where the next block of information is located. These chain addresses, supplied by programming, are used by the hardware to advance automatically to the next character location.

   Because storage is not infinite, it is possible to place a special indicator in the chain-address location of any block. When this buffer-block signal is detected by the machine in the process of acquiring a new block, automatic entry into the attention mode occurs, thus enabling the program to accurately control the available storage pool.

CONCLUDING REMARKS

The Translator described performs a mapping of a source program to an equivalent, list-structured, internal form. This method may be called "selective" or "differential" compiling, because statements may be inserted, replaced, and deleted without retranslating the entire source program. In addition, this approach provides rapid, comprehensive reference and diagnostic data. And finally, the process is reversible; the source program may be regenerated in its original form, or in a related form.[32]

Interpretive execution provides the means for complete source-language debugging. Information on the dynamic behavior of data use and control flow can be applied to improve optimization of the generated object code.

The implementation and description of the remote-computing system has naturally been done in a time-sharing context. Nevertheless, the techniques used are equally applicable to a conventional compiler operating under a monitor system.

Standard hardware devices in a conventional configuration were adapted to this purpose through programming. However, system performance could be substantially improved by use of a special machine organization designed to perform the same functions.

ACKNOWLEDGEMENTS

The authors wish to acknowledge the contribution of two associates: Miss Harriett Cohen for the storage allocation routines, and Mr. Dan Davis for the I/O list decomposition and interpreter routines.

APPENDIX I—EXPRESSION DECOMPOSITION/RECOMPOSITION

Introduction

The primary purpose of any formula translator is to reduce expressions to a form that provides the fundamental order in which operations should be performed to produce correct results. Implicit in this form should be a record of the order in which partial results are developed, accumulated, and reused.

The techniques and traditional programming tools generally applied to accomplish this are:[33, 34, 35, 36]

1. Forward scan;
2. Push-down list;
3. Forcing tables;
4. Ordered macro list;
5. Implied push-down temporary indications;
6. Chaining and string concatenation.

Forcing tables are used to produce an order of operation based on the real or assumed hierarchy of arithmetical or mathematical operators. Push-down lists in this respect often work on a LIFO (last in-first out) principle. Macros are used as a form which approaches as nearly to a machine-executable form as can be used while retaining its machine-independent structure. In addition to the operator and the operand elements, the macro form often

contains a reference to the temporary result that the operation will produce, such as T1 or T2, implying a push-down order to partial results. String-manipulation techniques of chaining and concatenation are often employed to facilitate translating operations.

Requirements

The Arithmetic Translator includes not only the traditional decomposition but also a recomposition[37] phase to restore the statement or expression to its original form from the compressed macro string generated during decomposition. The macro string generated, therefore, must satisfy several requirements:

1. It must be easily interpreted, saving time;
2. It must be compact, saving space;
3. It must be recomposable.

The decomposition translator must detect all errors in logic and syntax. It must supply the number, order, and mode of all operations to be performed by the Interpreter.

The recomposition translator should develop a string in the original sequence; all necessary punctuation must be restored. In short, it must produce a string identical in all respects to the original, except for the removal of redundant parentheses. The resulting string, when decomposed again, should produce a macro string identical to that originally decomposed.

OPERATOR		DECOMPOSITION		RECOMPOSITION	
Symbol	Name	Left Op	Right Op	Old Op	New Op
+	plus	5	5	2	1
-	minus	5	5	2	1
*	multiply	4	4	3	2
/	divide	4	4	3	2
**	exponentiation	4	3	3	3
so	subscript	6	0	7	1
fo	function	6	0	7	1
=	replacement	7	0	7	1
um	unary minus	5	1	2	1
,	comma	6	5	0	1
(	left paranthesis	6	0	0	0
)	right paranthesis	0	6	0	0
⊙	end of message (EOM)	0	7	0	1

Note: The zero code signifies that the operator is illegal when appearing in the specified role.

Figure 2.8. Forcing Tables for Translator.

Techniques

Forcing Tables

The forcing tables in Figure 2.8 are used as follows. The decomposition table is used to cause the generation of macros based on the relative hierarchy of related or successive operators. If the value for the right operator is equal to or greater than the value for the left operator, then a macro based on the left operator is generated.

The recomposition table is used to decide when parentheses are necessary to maintain the hierarchy implicit in the order of macros previously generated. If the value for the new operator is greater than or equal to the value for the old operator, then the string developed around the old operator during a previous concatenation must be enclosed within parentheses before further concatenation can take place.

Push-Down Lists

The lists used in the decomposition translator are the operator and variable lists which hold those elements awaiting further action from a forcing situation. The recomposition translator has an operator list used essentially for the same purpose. In addition, it uses two lists which contain control words of partial strings awaiting further action. The "work list" contains the control words of strings which are to be concatenated into a single string with a single control word. This control word is then placed on the "string list" until a later call for further concatenation is encountered.

Macro Strings

Each macro contains an operator byte and one or two variable bytes. The operator is a basic operation plus an indication of the modes of the variable bytes. Either or both variable bytes may contain a temporary indication. These do not have to be specific temporary indications, since owing to the ordered structure of the macros, both the Interpreter and the recomposition translator use a push-down accumulator for storing and fetching partial results of execution and partial strings developed through concatenation. For the same reason, no indication need be kept in the macro of the

temporary to be generated by the operator (such as T1 or T2). To save space, macros with temporary indications may be compressed during packing procedures by indicating left and/or right temporaries in the operator byte, thus eliminating all bytes for temporary indications. For example, when the macro string generated in the example is compressed in this manner, it results in 22 bytes, or, at four eight-bit bytes per word, less than six words of storage. The Interpreter accesses macros—in order—for execution of the statement, building up temporary results and using them in turn when later macros call for them. An EOM (End of Message) operator signals the end of the macro list. The recomposition translator accesses the macros and builds up temporary strings in much the same manner as the Interpreter. Also from this simplified, compact macro form it is but a simple step to generate machine-language code; either temporary locations can be implicitly addressed by the machine itself or else explicit storage addresses can be used.

Chaining and Concatenation

In the recomposition translator, when an operand is not an intermediate temporary, it is developed as an element in the output string and placed in an empty word in a pool. It is treated as a one-element string and assigned a control word. When strings are to be joined together, the last word of the first string refers to the linking operator (which is developed in the empty pool), and in turn, the operator refers to the first word of the next or preceding string. The two or more control words are combined into one which references the first and last words of the concatenated chain or string of elements. When the recomposition translator eventually encounters the EOM operator, there is only one chain represented by a control word on the string list. This chain or scrambled string is then unraveled into a sequential list of all the elements in the recomposed statement or expression.

	i.	Mixed Mode
	ii.	Mixed Mode in a Function Argument
	iii.	Illegal Use of Function or Array Name Without Arguments
*	iv.	Simple Variable, Constant, or Expression followed by left Parentheses
	v.	Illegal Mode of Function Argument
	vi.	Illegal Number of Arguments in Function
	vii.	Fixed to Float Exponent
	viii.	Level of Nesting of Functions Exceeds Maximum Number of Eight (8)
*	ix.	Illegal Successive Operators
*	x.	Illegal Parenthetical Order
*	xi.	Uneven Number of Parentheses
*	xii.	General Syntax Error
*	xiii.	Expression Begins with Illegal Operator
	xiv.	Mode of Variable Not Set
	xv.	Mode Not Set For Any Arguments in Function
	xvi.	Number of Parameters in Function Exceeds Declared Maximum
	xvii.	Number of Arguments in (Defined) Function Specified to be Zero
	xviii.	Mode of Actual Argument is Not Set
*	xix.	Illegal Operator in Parameter or Illegal Position for Comma
	xx.	A**B**C condition - (illegal in FORTRAN)

* Only these errors cause an immediate error return.
All others return for further error checking.

Figure 2.9. Arithmetic Translator Diagnostics.

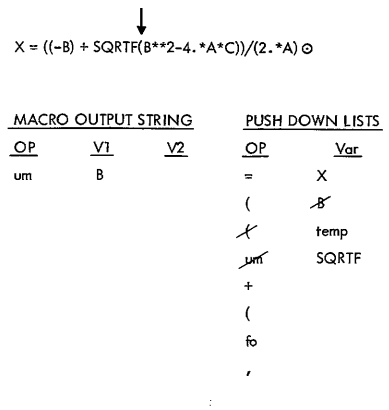


Figure 2.10a. Example of Decomposition—Part I.

Diagnostics

The decomposition performs complete diagnostic checking. Wherever possible, error checking continues even though some errors

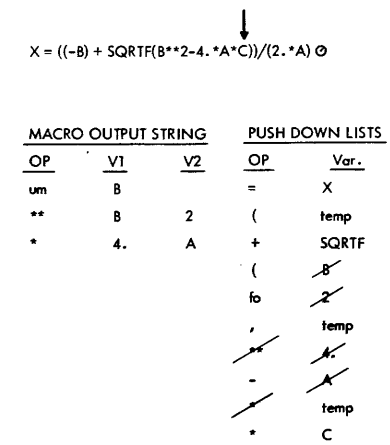


Figure 2.10b. Example of Decomposition—Part II.

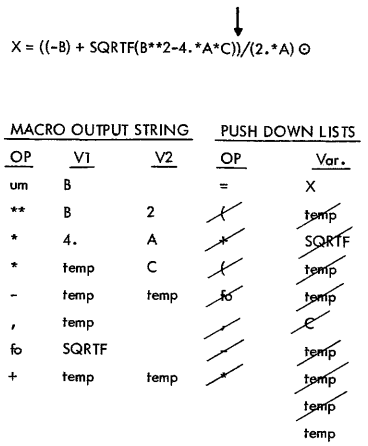


Figure 2.10c. Example of Decomposition—Part III.

have already been encountered. (See Figure 2.9 for a list of decomposition diagnostics.)

Decomposition Rules (Figure 2.10)

1. When all action has been taken with a new operator or variable encountered in the forward scan, it is placed on the appropriate push-down list.
2. An array name or function name followed by a left parenthesis generates two additional operators for the operator list: a subscript operator or function operator, and a comma operator for the initial parameter.
3. When the forcing value of a new operator equals or exceeds the forcing value of the last operator on the operator list, action is taken to output a two- or three-byte macro:
   a. The last operator on the operator list and the last one or two variables on the variable list, depending on the operator, are removed and incorporated into an output macro.
   b. For each macro generated for the output string, except for comma-operator macros, a temporary indication is generated on the variable list.

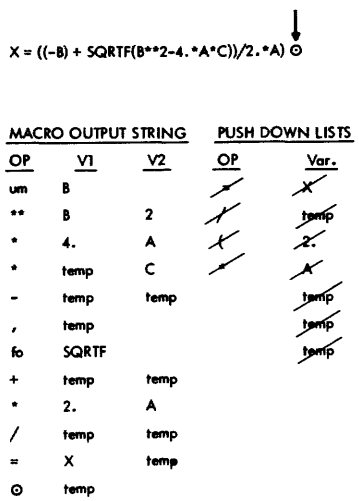


Figure 2.10d. Example of Decomposition—Part IV.

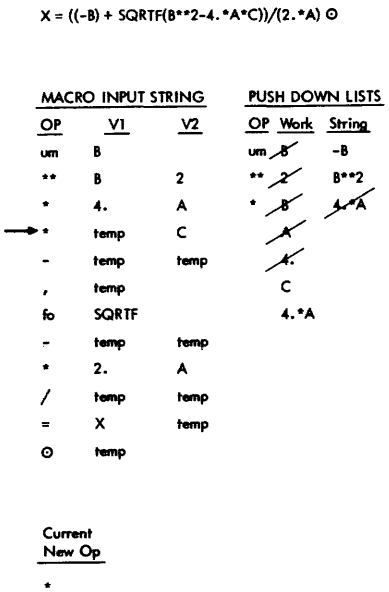


Figure 2.11a. Example of Recomposition—Part I.

Recomposition Rules (Figure 2.11)

1. On each new macro encountered in the forward scan, the right operand (V2), if it exists, is always considered for action before the left operand (V1).
   a. If V_i of a macro is not a temporary indication, it is developed in a word from the empty pool, and assigned to a control word which is placed on the intermediate work list.
   b. If V_i of a macro is a temporary indication, the last control word on the string list is removed and placed on the work list.
2. For any operator in a macro except the comma operator, the last action taken is to

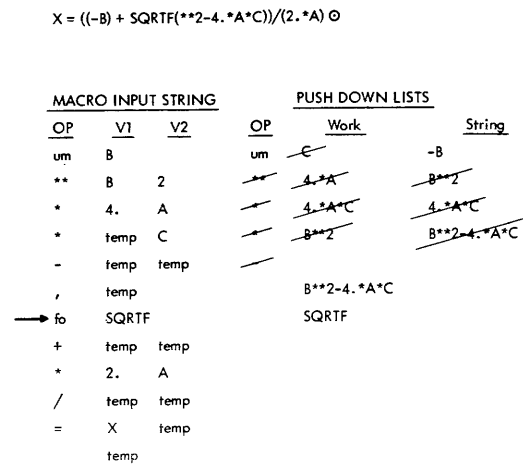


Figure 2.11b. Example of Recomposition—Part II.

place the operator temporarily on the operator push-down list, and to combine the control words on the work list, linking their strings together into one control word, which is placed on the string list.

a. For subscript and function operators, the name and parameter strings referenced on the push-down work list are linked in order, separated by appropriate parentheses and commas.

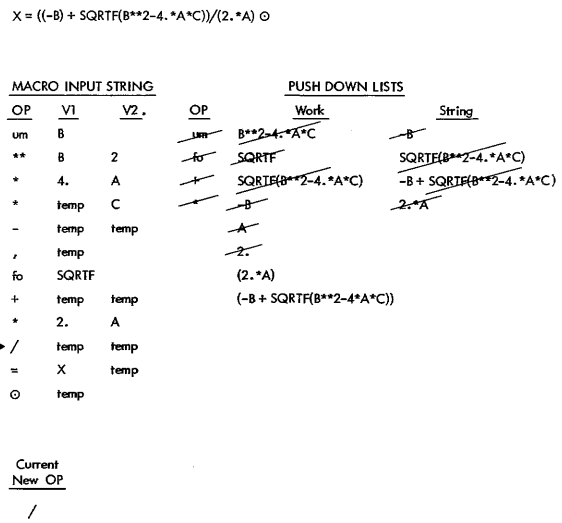


Figure 2.11c. Example of Recomposition—Part III.

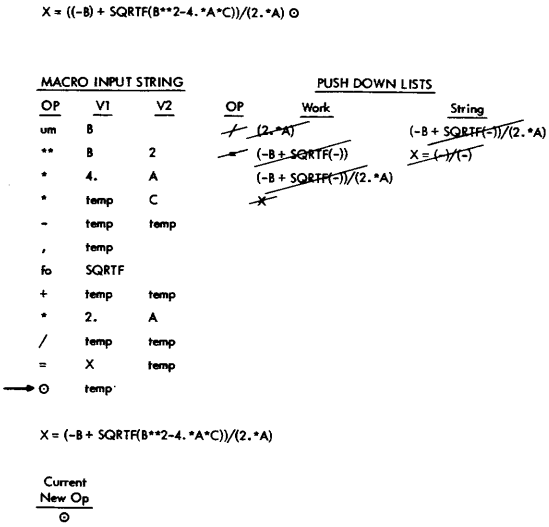


Figure 2.11d. Example of Recomposition—Part IV.

b. For arithmetic unary or binary operators, the one or two strings referenced on the work list are linked with the operator.

3. Whenever a control word is removed from the string list, an operator is removed from the operator list and tested against the new operator from the current macro.
   a. When the right forcing-value of the new operator equals or exceeds the left forcing-value of this last operator from the operator list, parentheses are placed at the ends of the string referenced by the control word just placed on the work list.
   b. If the string represents V2, and the left forcing-values of the new and last operators are equivalent, parentheses are placed at the ends of the string.

Extensions

There is no limit to the length of the statement string that can be used as input to this type of decomposition translator.

Any mathematical language based on hierarchical rules of operation—for purposes of computation similar to that in arithmetic formulas—can be decomposed and recomposed just as easily using forcing tables and the other traditional techniques. The macro form pro-

duced could be of quite a different form depending upon the nature of the interpretive scan. It would, of course, have the same implied order of operation. The operators involved need not be only unary or binary operators; they need not be only arithemtical or functional. Boolean operators, logical operators, ternary operators, or any others could be easily handled in this manner.

REFERENCES

1. A. NEWELL (Ed.), "Information Processing Language—V Manual," Prentice-Hall, 1961.
2. H. GERLERNTER, J. HANSEN, and C. GERBERICH, "A FORTRAN-Compiled List-Processing Language," *ACM Journal*, April 1960.
3. A. J. PERLIS and C. THORNTON, "Symbol Manipulation by Threaded Lists," *ACM Communications*, April 1960.
4. A. EVANS, A. PERLIS, and H. VAN ZOEREN, "The Use of Threaded Lists in Constructing a Combined ALGOL and Machine-Like Assembly Processor," *ACM Communications*, January 1961.
5. J. WEIZENBAUM, "Knotted List Structures," *ACM Communications*, March 1962.
6. J. WEIZENBAUM, "Symmetric List Processor," *ACM Communications*, September 1963.
7. R. BROOKER and D. MORRIS, "A General Translation Program for Phrase Structure Languages (Lists)," *ACM Journal*, January 1962.
8. H. W. LAWSON, "The Use of Chain List Matrices for the Analysis of COBOL Data Structures," *Proc. ACM National Conference*, September 1962.
9. H. D. BAECKER, "Mapped List Structures," *ACM Communications*, August 1963.
10. P. R. KOSINSKI, H. KANNER, and C. L. ROBINSON, "A Tree-Structured Symbol Table for an ALGOL Compiler," *Proc. of ACM National Conference*, August 1963.
11. P. M. SHERMAN, "Table Look-at Techniques," *ACM Communications*, April 1961.
12. L. R. TURNER, A. MANOS, and N. LANDIS, Initial Experience on Multiprogramming on the Lewis Research Center 1103 Computer," *Proc. of ACM National Conference*, August 1960.
13. N. LANDIS, A. MANOS, and L. R. TURNER, "Initial Experience with an Operating Multiprogramming System," *ACM Communications*, May 1962.
14. A. B. SHAFRITZ, A. E. MILLER, and K. ROSE, "Multi-level Programming for a Real-Time System," *Proc. EJCC*, December 1961.
15. E. F. CODD, "Multiprogram Scheduling," *ACM Communications*, June 1960, July 1960.
16. E. F. CODD, "Experience with a Multiprogram Scheduling Algorithm," *Proc. of ACM National Conference*, August 1960.
17. J. WEGSTEIN, "From Formulas to Computer Oriented Language," *ACM Communications*, March 1959.
18. B. ARDEN and R. GRAHAM, "On GAT and the Construction of Translators," *ACM Communications*, July 1959.
19. M. E. CONWAY, "Design of a Separable Transition Diagram Compiler," *ACM Communications*, July 1963.
20. The 701 Speedcoding System, IBM Form Number 24–6059.
21. The 705 Print System, IBM Form Number 32–7855, 1957.
22. Bendix Intercom 1000 Programming System, Bendix Computer Division, 1958.
23. W. R. BRITTENHAM, et al., "SALE (Simple Algebraic Language for Engineers)," *ACM Communications*, October 1959.
24. R. E. MACHOL, W. J. ECCLES, and J. C. BAYS, "There's Still a Place for Interpreters," *Proc. ACM National Conference*, September 1962.
25. W. LONGERAN and P. KING, "Design of the Burroughs B5000 System," *Datamation*, April 1961.
26. R. W. BARTON, "A New Approach to the Function Design of a Digital Computer," *Proc. WJCC*, April 1961.
27. J. ANDERSON, "A Computer for Direct Execution of Algorithmic Languages," *Proc. EJCC*, December 1961.

28. A. C. D. HALEY, "The KDF. 9 Computer System," *Proc. FJCC,* December 1962.
29. C. B. CARLSON, "The Mechanization of a Push-Down Stack," *Proc. FJCC,* November 1963.
30. 7740 Communications Control System Principles of Operation, IBM Form Number A22–6753.
31. 7740 Communications Control System Communications Control Package, IBM Form Number C28–8160.
32. J. J. ALLEN, D. P. MOORE, and H. P. ROGOWAY, "SHARE Internal FORTRAN Translator (SIFT)," *Datamation,* March 1963.
33. K. SAMUELSON and F. L. BAUER, "Sequential Formula Translation," *ACM Communications,* February 1960.
34. H. D. BAECKER, "Implementing a Stack," *ACM Communications,* October 1962.
35. C. L. HAMBLIN, "Translation to and from Polish Notation," *The Computer Journal,* October 1962.
36. R. J. EVEY, "Application of Pushdown-Store Machines," *Proc. FJCC,* November 1963.
37. K. IVERSON, "A Programming Language" (Chapter 5), John Wiley & Sons, Inc., 1962.

MULTICOMPUTER PROGRAMMING FOR A LARGE SCALE REAL-TIME DATA PROCESSING SYSTEM

G. E. Pickering, E. G. Mutschler, and G. A. Erickson
UNIVAC Division of Sperry Rand Corporation
San Diego, California

INTRODUCTION

The multicomputer programming techniques discussed in this paper were conceived and implemented in a large scale tactical data system developed for the U.S. Navy. Many of the details of this system are classified and cannot be discussed here.* However, it can be stated that the subject system is a man/machine complex, primarily intended for fleet air defense and surface operations and maneuvering. The objectives of the system are: to provide commanders of forces afloat with a broad picture of the current tactical situation; to assist in directing operations in time to intercept and destroy potential threats; and to present the means to coordinate various weapon systems in a combat environment. These are achieved by automating, to a high degree, the collecting, processing, exchanging, and evaluating of large quantities of data through use of computers and digital data processing techniques.

The system objectives are accomplished in real-time. Data are received by the system from various sensing devices such as shipboard and AEW radars, sonar, IFF equipment and ECM equipment which are in continuous contact with the outside environment. Data entering the system are processed, analyzed and used by the system *to influence or alter an event during the progress of that event,* i.e., real-time.

*This paper was originally prepared for presentation at the 1961 Eastern Joint Computer Conference. Security regulations prevented its release until now.

Fundamental in the design philosophy of this system is the Unit Computer concept. In essence, the concept is a recognition of the need for computers of varying capability among ships of various types and operating modes. It solves this requirement by the use of standard computers operating in multiples to obtain increased capacity and functional capability, rather than use of several different computers each possibly of a different shape and size. However, in solving this varying computer capacity requirement, the Unit Computer concept established the need for multicomputer programs.

This paper concerns the programming problems encountered in designing and implementing operational programs requiring more than one Unit Computer and describes the techniques developed to solve two intrinsic problem areas:

- EXECUTIVE CONTROL IN A MULTICOMPUTER COMPLEX
- DATA TRANSFER BETWEEN COMPUTERS

A third problem is assignment and distribution of tasks between computers. To deal with this subject in the detail required would necessitate discussing classified material. That this is a problem of complexity should be obvious. However, it is evident that distribution of tasks between computers is a system design problem which must be uniquely solved for each multicomputer system.

It should be emphasized that the multicomputer programming techniques discussed in this paper have been tried and tested for the last five years. Several multicomputer programs, employing two and three computers, have been delivered to the Navy. Thus, multicomputer programming, as discussed in this paper, is not theory, but actuality.

In preparation for the subject of multicomputer programming, it is first necessary to discuss pertinent characteristics of the Unit Computer developed by UNIVAC to satisfy the Unit Computer design philosophy.

I. THE UNIT COMPUTER

The Unit Computer is a general-purpose, stored program, solid-state machine.* Only those characteristics most pertinent to the subject of multicomputer programming are discussed in this paper.

Real-time Clock

Among the features that make possible the use of the computer in real-time applications is the real-time clock. The clock is contained in a special register within Magnetic Core Memory and can be referenced or set by the computer program. Time is maintained accurate to the nearest 2^{-10} second. The clock is incremented automatically by the computer upon signal of a 1/1024 cps crystal controlled oscillator.

When the system is started, the clock is set to zero and time is automatically maintained relative to the start of the system problem. Although the modulus of the clock is approximately 7 days, the real-time modulus of the system may be somewhat less. The real-time modulus of the system is so defined that in scheduling tasks for execution (see discussion under Section II—Program Control in a Multicomputer Complex), the modulus of the clock will not be exceeded.

In a multicomputer program, the real-time clocks within all computers are synchronized. Upon initiation one computer transmits the content of its real-time clock to the other computers, which then set their clocks accordingly.

DATA TRANSFER LOGIC

A total of 14 input and 14 output channels is provided in the Unit Computer. Of these, two input and two output channels are especially designed for intercomputer communications. Each input channel and output channel consists of 30 data lines and 3 control lines. The special input/output channels differ from a normal input/output channel primarily in use of the control lines and in the associated channel logic.

The input/output logic of the Unit Computer requires that peripheral equipment request each input or output data word transfer. The computer then *acknowledges* receipt of the input or availability of an output. All input channels, including the special input channels for intercomputer transfer, and all normal output channels use Request and Acknowledge logic.

The control lines associated with request and acknowledge logic are shown in Figure 1. These are Input Data Request Line and Input Acknowledge Line for normal input channels and

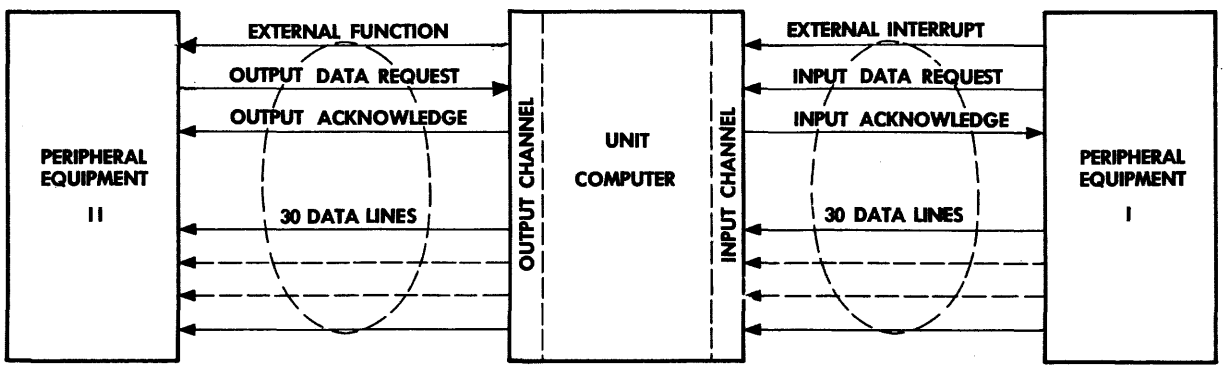


Figure 1. Normal Channel Configuration.

*A more thorough description can be found in "Datamation," December 1961, p. 45.

Output Data Request Line and Output Acknowledge Line for normal output channels. In addition to these two lines normal input channels have an External Interrupt control line and normal output channels have an External Function control line. The External Interrupt control line allows the external equipment connected to that channel to signal the computer of its immediate data transfer requirement which is honored by an interruption of the computer program. The External Function control line is used to specify a function desired of the external equipment connected to that output channel. An External Function Word to a tape control unit, for example, may specify *Rewind Tape Unit 2.*

The two special output channels use Ready and Resume logic for intercomputer data transfer control. The transmitting computer signals that it is Ready with an output data word which is interpreted by the receiving computer as an Input Data Request signal. The receiving computer upon accepting the transfer will send back an Input Acknowledge signal which is interpreted by the transmitting computer as a Resume. This reverse logic is necessary so that the transmitting computer will exhibit to the receiving computer the same control and timing characteristics as do the peripheral equipment.

The control lines associated with *Ready* and *Resume* logic of the intercomputer channels are illustrated in Figure 2. These are Input Data Request (or Ready) and Input Acknowledge (or Resume). In addition, intercomputer channels utilize an Input Buffer Status signal. The Input Buffer Status signal originates in the receiving computer and is defined by the state of the Input Buffer Active/Inactive Designator. The transmitting computer can sense the Input Buffer Status signal, thus determining status of the receiving computer's input buffer on that intercomputer channel.

A special output register (C1), incorporated in the computer, alternately services the two output channels reserved for intercomputer communication. All output intercomputer communications are time-shared through this special output register which holds data until a Resume signal is received from the receiving computer. However, if a Resume signal is not received within 32 to 64 seconds, the register is automatically cleared and the computer is notified of this condition by an Internal Interrupt called the *Intercomputer Failure Monitor.* This 32- to 64-second time interval is necessary to insure that intercomputer output with other computers is not suspended indefinitely. The Interrupt utilizes special entrance registers in Magnetic Core Memory for notification of faulty intercomputer output channels and to distinguish which of the channels is faulty.

Data transfer in or out of the Unit Computer, whether between computers or between computer and external equipment, is handled by buffered transmission of data with timing under control of the receiving computer or the external equipment respectively. The buffering process transfers consecutive words, starting at a given initial address through a given terminal address, on a specified input or output channel. A single computer instruction will initiate a buffer mode of data transfer. Once established, buffer transmission employes independent access to memory; the entire buffering operation proceeds to completion with no additional program references. Thus, the buffer mode of data exchange provides for input/output operating asynchronously with the main computer program; the computer continues execution of program instructions in the normal sequence.

The sequence of data and control signals for a normal transfer of data from one computer to another would proceed as follows:

a) Receiving computer sets Input Buffer Active Signal;

b) Transmitting computer detects Input Buffer Active Signal;

c) Transmitting computer places data on 30 data lines;

d) Transmitting computer sets *Ready* which becomes Input Data Request in Receiving computer;

e) Receiving computer detects Input Data Request;

f) Receiving computer samples 30 data lines;

g) Receiving computer sets Input Acknowledge line (returned to Transmitting computer as *Resume*);

h) Transmitting computer senses Resume line;

i) Transmitting computer drops data lines and Ready line.

Steps c) through i) of this sequence are repeated for every data word. Input Buffer Active remains energized during the entire transfer of a block of words.

In most cases it is desirable for the program to be informed of the completion of a particular buffer transmission. This is made possible by *internal interrupts* generated within the computer. If an input buffer mode via a specified channel was established with a buffer monitor, an *Input Buffer Monitor Interrupt* is generated when the buffer mode terminates (i.e., transmission has been completed as indicated by the current buffer address equal to the terminal address). Likewise, an output buffer established with monitor would generate an *Output Buffer Monitor Interrupt* when the buffer mode terminates. A Buffer Monitor Interrupt is associated with each input channel and each output channel; a separate entrance register in computer memory is reserved for each Interrupt. The unique entrance address thus defines the source of the Internal Interrupt request for action by the appropriate Interrupt routine.

COMPUTER INSTRUCTIONS

In addition to instructions normal to arithmetic, logic, and index operations there are several instructions which add tremendously to the power of the computer in a real-time application. Those instructions which are mentioned or inferred later in the paper are now described.

Repeat—The Repeat instruction sets up a *repeat mode* whereby the instruction immediately following the Repeat instruction is executed n times, as specified in the Repeat instruction. In initiating the repeat mode, n is transferred to a special register which retains the number of executions remaining throughout the repeat mode. If n is zero, the instruction to be repeated is skipped. This instruction also enables automatic address modification (increment, decrement, or increased by a pre-set constant) of the repeated instruction after each individual execution.

Ordinarily the repeated instruction searches a table for coincidence. If coincidence does not occur, the repeat mode terminates and the instruction following the repeated instruction is executed. If coincidence occurs, the repeat mode terminates and the instruction following the repeated instruction is skipped. This method of searching a table for coincidence is used by the Executive Routine in controlling the execution of various tasks of the main program, as discussed later. A principal advantage of a repeated search is the reduction in instruction execution time since a memory reference is not required to read in a next instruction. As used in the Executive Routine, a repeat search takes $8 + 11.2$ (N) microseconds where N is the table item number on which coincidence occurs. A non-repeated search to accomplish the same end would require $8 + 40$ (N) microseconds. Thus, the time required for program control is reduced by an approximate factor of 3 through use of the Repeat instruction.

Compare—This instruction compares the signed value of the operand with the signed value contained in either or both the arithmetic registers, A and Q. Skipping the program's next sequential instruction is also allowed by this instruction if the Skip condition is met.

The Compare instruction is used with the Repeat instruction to perform the search described previously. The execution time given in that description is for a repeated Compare. As will be described later in Section II of this paper, the time at which a task is scheduled to be executed is compared with the contents of the real-time clock.

Return Jump—The Return Jump instruction provides for transferring control to a specific set of instructions (subroutine), executing that subroutine, and returning control to the next instruction following the Return Jump. The Return Jump makes possible the modular program construction used whereby each separately defined task is implemented by a subprogram, which is referenced by the Executive Routine, and the identical subprogram is used in each program performing that task. A subprogram is a high-level subroutine; a subprogram usually references lower-level subroutines.

External Function—The External Function instruction has two meanings, one for intercomputer channels and another for normal input/output channels. If the instruction specifies an intercomputer channel, the connected computer is interrogated as to the status of its input buffer on the connected channel. If the interconnected computer's input buffer is active (i.e., an input buffer has been initiated and the connected computer is ready to receive data), the next instruction following the External Function is skipped. If the connected computer's input buffer is not active, the next instruction (which is normally a Jump) is executed.

For normal input/output channels, the operand of the External Function instruction (normally a control code) is transmitted to the external equipment connected to that channel.

Store Input Channel—This instruction stores the contents of the specified input channel at the address specified by the operand. An Input Acknowledge signal is then sent over the specified channel thereby informing the external equipment of the computer's availability to receive additional data and to indicate that the previous input was received. Since buffer mode of data transfer is normally used, the Store Input Channel instruction is used primarily to generate Input Acknowledge signals to External Interrupts.

Initiate Input Buffer (With Monitor) and Initiate Output Buffer (With Monitor)—These instructions establish an input or output buffer, respectively, via the specified input (or output) channel. Subsequent transfers of data, executed at a rate determined by the external device, are made directly into (or occur directly from) Magnetic Core Storage starting at the address specified by the operand. The storage address initially established is advanced by one for each individual transfer. The next current address is maintained throughout the buffer process in the lower half word of the Buffer Control Register for the specified channel; the last address of the transfer is maintained in the upper half word of the same Buffer Control Register. Each input channel and each output channel has a unique Buffer Control Register associated with it. The buffer mode will continue until it is superseded by subsequent initiation of a new buffer via the same channel or until the upper and lower half words of the Buffer Control Register are equal. Should the latter occur, a Buffer Monitor Interrupt is generated, causing the computer program to be interrupted and the Buffer Monitor Interrupt routine for that channel to be executed.

Multicomputer programs use the Initiate Input Buffer (With Monitor) and Initiate Output Buffer (With Monitor) instructions exclusively in transmitting data between computers.

II. EXECUTIVE CONTROL PHILOSOPHY

The system which made necessary multicomputer programming techniques has a number of functions that must be performed. In accomplishing these functions, the design objective is to use both men and machines to best advantage. Repetitive and routine operations are performed automatically by machines, whereas decisions of tactical importance are made by men. Although the focal point of the equipment complex is one or more computers, the system also employs high-speed digital data communication facilities, radar video processors, analog-to-digital data converters, digital-to-analog converters, and typical computer peripheral equipment such as magnetic tape and Teletype,* all of which provide for automatic inputs to the computer and/or automatic outputs from the computer. A complete display of complex and data entry devices allow men to monitor an ever-changing, yet accurate picture of the current tactical situation and to enter their intelligence and tactical decisions into the computer.

All equipment is connected directly or indirectly to the computers. For each function the computers must:

- receive data from the appropriate system equipment, consistent with data transfer rates of each equipment;
- correlate, process and (as necessary) evaluate data in performing the specified system function;
- present the results of processing and evaluation to men for their interpretation and decisions;

*Teletype is a trade mark of the Teletype Corporation.

transmit pertinent data and human decisions, as appropriate, regarding that data to the ultimate users, again consistent with data transfer rates of the equipment involved.

The system operates in real-time, therefore, operational requirements dictate the response times for individual system functions.

To relate a real-time system to real-time data processing then, one might say that real-time data processing is processing done immediately as a result of an input for the purpose of providing an essentially instantaneous response or output.* Processing must, as a result, be completed at a rate greater than the input data rate or else the processing will fall behind so that eventually the system will be operating out of real-time. The system saturation point is reached when processing rate equals the input rate.

From the foregoing discussion, it is clear that any real-time system must have, at least, a data processing capability sufficient to handle the average input rate from all input functions. Moreover, since system inputs occur randomly, it must be assumed that all inputs could occur simultaneously. This would represent the peak load. The problem then, is how to schedule, or queue, the functions to be performed in such a way that under peak load conditions all functions would be performed within their real-time requirements.

An Executive Control Philosophy has been developed and implemented to solve the scheduling (or multiprogramming) problem. Executive Control Philosophy is a general term—explained specifically it means a recognition of the system's tasks in order of system priority; distributing them among the system's computers; and then controlling them in the individual computers of the system by an Executive Routine within each computer.

The Executive Control Philosophy is based on the fact that each function and concomitant handling of peripheral equipment defines a separate task (or set of tasks) that must be performed by the computers.

Each system task is performed by a computer subroutine (or group of routines) called a "subprogram." By definition, a subprogram is any subroutine referenced directly by the Executive Routine.

An Executive Routine is contained in each computer of a multicomputer program and maintains complete control of tasks assigned to that computer. *Over-all control of a multicomputer program by a Master Executive Routine is not employed.* Rather a method of reciprocal control is used. One of the system tasks (i.e., subprograms) within each computer in a multicomputer system is concerned with intercomputer data transfer. The function of each intercomputer subprogram is to process the data received from an interconnected computer and upon completion of this processing to initiate the return data transfer. To effect reciprocal control, program logic must be such that the residual state of intercomputer transfer within each computer is input. Thus, the computer which has control of the intercomputer data transfer (i.e., next to initiate an output buffer transfer) actually has temporary control of the two-computer operation. Reciprocal control, therefore, means that control is assumed by one, then the other, of two interconnected computers. In systems utilizing three or more computers, reciprocal control would be used for each pair of interconnected computers. Details of intercomputer data transfer are discussed in Section III.

Subprograms within a computer are considered for execution on the basis of their priorities as system components. System priority for subprograms is relative, and is determined in the following manner: If tasks corresponding to subprograms A and B have response requirements of 100-milliseconds and one second, respectively, then subprogram A would have a higher system priority than subprogram B. System priorities for all other subprograms (or groups of subprograms associated with one system function) are chosen similarly.

Normally, tasks associated with processing inputs to the computer from system equipment (and outputs where stringent timing require-

*Discussions of the concept of real-time can be found in "Computers and Automation"—September, 1963, p. 26, and "Datamation"May and June, 1963, pp. 29 and 28 respectively.

ments are imposed by external equipment) have highest priority. This is particularly true if failure to accept inputs when available means loss of important, non-repeated data and possible system malfunction. However, in some cases where inputs are under more positive control of the computer or of a repetitive nature it is possible to lower the priority of equipment handling subprograms to insure processing of previously received data before new inputs are accepted. This has the advantage of effectively shutting off or delaying inputs during brief periods of saturation. Consider for example two subprograms associated with the display function: one subprogram periodically interrogates displays for manual action requests and the other processes these requests and generates the proper response. By assigning the processing subprogram a higher system priority than the interrogation subprogram, the rate of processing will automatically control the interrogation rate. Only under extreme saturation for a prolonged period would the operator even notice the delay; thus under normal operation the system response time for the display function would be honored.

For the reasons given above—to protect against loss of important non-repeated data and to insure processing of previously received data within the real-time response requirements—intercomputer subprograms are assigned a relatively high priority. Conversely, each computer might contain a subprogram of lowest priority to perform such operations as program checking when no other system task required execution.

Consider a list of all subprograms ordered by priority. The Executive Control Philosophy, applied in an elementary form, would dictate that this list be scanned sequentially, executing subprograms when they are needed. After each subprogram execution, the scanning process would be resumed starting with the highest priority entry. Thus, under a peak load condition where all subprograms may require execution, the top priority subprogram would be executed instantaneously, the next highest immediately following, and so on. The resultant response time of each subprogram would be the sum of previous execution times for all higher priority subprograms, recognizing that some of the higher priority subprograms may be executed more than once. The assumption must be made that sufficient computational capability exists to perform all assigned tasks. If this is not true then complete penetration of the list will not be realized and the only solution is additional computers. If computational capability is sufficient to handle the average input rate for all functions, then under peak load conditions higher priority tasks are performed first and lower priority tasks are delayed until time is available. Thus, a lower priority task may be defined as not having a critical response time so that late execution will not seriously degrade the system. An example of such a task is output to displays which might need to occur at a rate of 15 times a second to maintain a flicker-free presentation. However, during peak load, which probably would last for less than a second, reducing the output rate to 14, 13, or less would normally go unnoticed.

As stated earlier, the real-time requirements of any system task are such that processing is done within the required response time. The Executive Control Philosophy takes advantage of this characteristic by providing the option of assigning a variable priority to a subprogram corresponding to a particular task. Consider again the subprogram corresponding to task A. Subprogram A could be executed upon demand as a top priority item, thereby completely satisfying the response requirements. But recall that the maximum response time tolerable was 100 milliseconds. A response from subprogram A would therefore be acceptable as late as 100 milliseconds after the original demand. Since the present computer load may be quite high, it might be desirable to defer execution of subprogram A for, say, 50 to 75 milliseconds. This is done in the same priority list described earlier, except that a time element is included in the ordering of the subprograms. Multiple entries for subprogram A can be made in the priority list as follows: An entry is made as low priority item for execution immediately, as an intermediate priority item to be executed in 50 milliseconds, and as a top priority entry to be executed in 75 milliseconds. Thus, after a processing demand has been received, subprogram A will be executed:

a) immediately out of the low priority entry, if the computer load is low;

b) out of the intermediate priority entry in 50 milliseconds, if the computer load is moderate; and

c) out of the top priority entry in 75 milliseconds, under peak load conditions.

This assures that under all load conditions subprogram A will be performed within the required response interval. It should be noted that the above procedure will result in automatic "smoothing" of the processing load, and will still allow execution of each subprogram in the absolute minimum time, consistent with system priority and existing computational load. A subprogram is executed only when its "flag" is set. By definition, a flag is set when the time at which the task is scheduled to be executed is equal to real-time.

2) When the Executive Routine detects that a flag is set, it reschedules the time when the next flag for this subprogram will be automatically set. This mechanism allows for tasks that must be executed at an approximate periodic rate, as mentioned earlier for output rate to displays.

3) One subprogram may set or partially set the flag of another subprogram. This mechanism is used when it is either necessary or desirable to execute certain associated subprograms in a definite sequence. For example, a subprogram which processes data placed in computer stores would be flagged by another subprogram which acquires that data through interrogation of an external equipment. Extending this same example by adding an additional processing subprogram, it may be necessary to complete both processing subprograms before reinitiating the interrogation subpro-

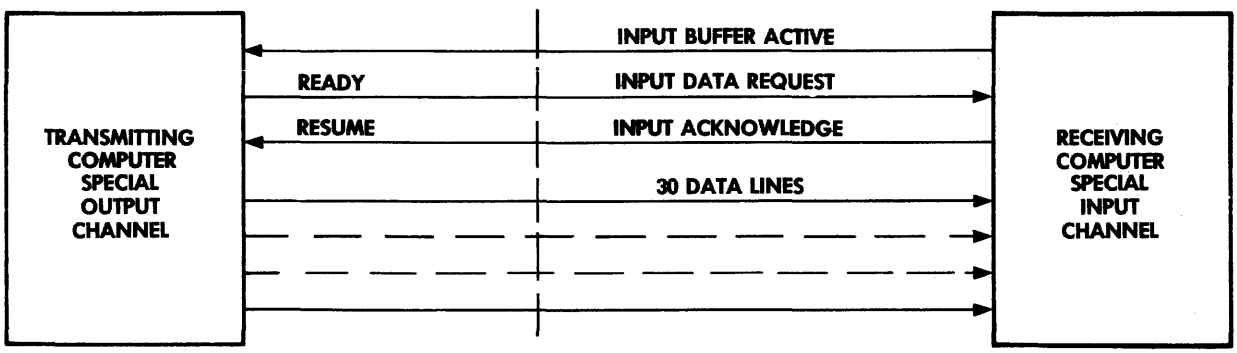


Figure 2. Special Channel Configuration.

Four mechanisms are provided to set these flags. Each subprogram normally uses but one of these mechanisms.

1) External Interrupts or Internal Buffer Monitor Interrupts set the flags of subprograms associated with servicing external equipment and most data transfers. Consider the task of transmitting data over a communication link. Communication terminal equipment would send an External Interrupt to the computer thereby signaling the start of a transmission period. The computer must have data to be transmitted available to the communication equipment within a certain time period. If the communication link is not operating, there is no need to execute the communication transmit subprogram.

gram. Since interrogation is periodic it would automatically be rescheduled by mechanism 2, above. However, a lockout (significant time period exceeding the real-time modulus of the system) could be added to the periodic interval; each processing subprogram when completed would partially set the flag by subtracting part of the lockout time. Thus, when both subprograms have been executed, the lockout is removed and interrogation will be repeated as per the periodic interval.

4) A particular subprogram may reset its own flag if, after receiving control, it determines that more than an allowable amount of processing time is required to complete the task. This mechanism is utilized only by lower priority subprograms and is designed to guar-

n	EXECUTIVE FLAG TABLE (EFT)	EXECUTIVE TIME TABLE (ETT)	EXECUTIVE JUMP TABLE (EJT) (sub-program)
1		24 seconds	A
2		24 hours	B
3		24 hours	C
4		10 seconds	D
5		24 hours	A
6		1 second	E
7		24 hours	B
8		50 millisec. + lockout	F
9		24 hours	G
10		24 hours	H
11		24 hours	I
12		24 hours	A
13		1 minute	J
M = 14		30 seconds	K

NOTE: 1. Subprograms D, E, F, J, and K are of a periodic nature.

2. Entries in the ETT are based on a 24 hour real-time modulus of the system.

3. The 24 hour entry in ETT represents a very large delay. This makes sure the flag is cleared automatically after the first reference.

Figure 3. Example of Executive Table Structure.

antee meeting the more stringent response time requirements of higher priority subprograms. When the Executive Routine again considers this task for execution on the basis of priority, control will be returned to enable further processing. The cycle will continue until all processing of this task is completed.

The Executive Control Philosophy is implemented through a unique method of table control. Three tables—Executive Flag Table (EFT), Executive Time Table (ETT), and Executive Jump Table (EJT)—contain entries for each assigned subprogram which are ordered by system priority. Each computer in a multicomputer program would have its own set of three tables with the subprogram entries unique to that computer. Figure 3 illustrates the one-to-one correspondence between the tables and also illustrates the use of a variable priority and lockout technique discussed earlier.

For a given subprogram, (n), EFT_n contains the time (per real-time clock) at which the subprogram is to be executed, ETT_n the automatic delay before the subprogram is to be repeated, and EJT_n the address of the subprogram. If the subprogram is to be executed at other than a periodic rate, the content of ETT_n is a sufficiently large number (usually the real-time modulus of the system) so that the task will not be repeated until one of the other flagging mechanisms resets the flag. In operation, the Executive Routine sequentially compares the computer's real-time clock against the entries in the Executive Flag Table, starting with the task of highest priority. In the case of a "hit" (the clock is greater than or equal to EFT_n), the corresponding Executive Time Table entry is added to current time, the sum is stored in EFT, and control is transferred to the corresponding subprogram as specified in EJT.

If the "hit" corresponded to a multiple entry subprogram, all EFT entries corresponding to that subprogram would be reset. When control is returned to the Executive Routine, or if no hit occurs, the search of EFT is repeated, starting again with the task of highest priority. Figure 4 is a flow diagram of Executive Routine logic.

The description of Executive Routine operation, as discussed in this paper, is short—but complete. Main control loop of the Executive Routine consists of eight instructions, one of which is the powerful *Repeat* instruction used in the search operation. The efficiency of a repeated *Compare* and the extremely small number of additional instructions necessary insure that only a minimum amount of computational time is usurped for the important area of program control.

III. DATA TRANSFER BETWEEN COMPUTERS

To accomplish intercomputer communications, stringent control must be exercised in establishing and terminating intercomputer buffers in each computer, thus insuring that data transfer between interconnected com-

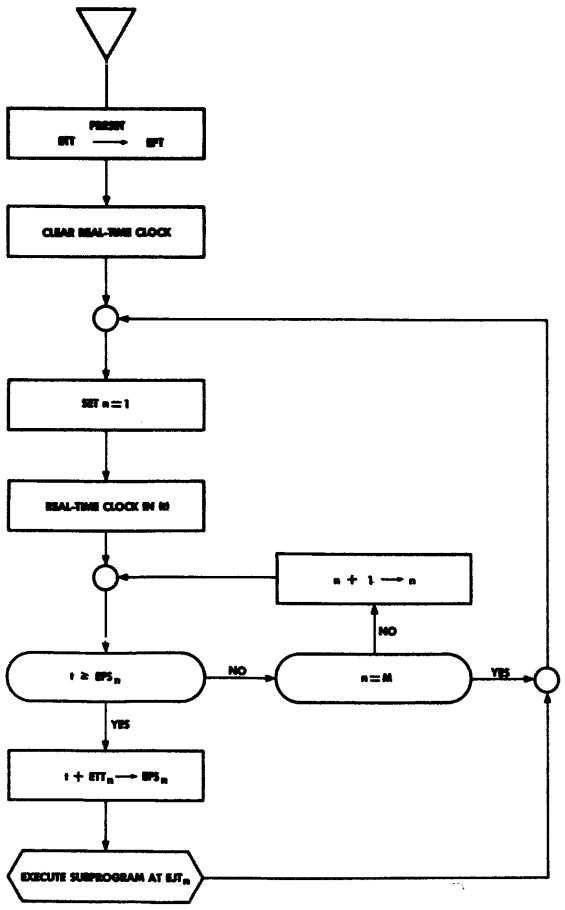


Figure 4. Executive Control Program.

puters will stay synchronized. Interlocks and check points for intercomputer communication are required in the hardware and in the programs. Special characteristics of the computer are combined with the program to foment this multicomputer concept.

As can be seen from the delineation of hardware characteristics in Section I, significant and powerful means are available to produce an efficient program for intercomputer communication. Internal Interrupts provide excellent program efficiency, in that they obviate periodic sampling requirements for detecting "end of buffer." Usually, within microseconds of buffer termination, the main program is interrupted to permit execution of program control tasks required by buffer termination.

In the following paragraphs, program characteristics and logic will be discussed to show the program's use of the computer's capability for data transfer.

PROGRAM CHARACTERISTICS

Elements of the logic governing the design of intercomputer programs are *variable length buffers and reciprocal control.* Fundamental to obtaining variable length buffers are the formats—message and buffer—and the method of controlling buffers. Reciprocal control means that control of the buffer operation is assumed by one, then the other, of two interconnected computers. In accomplishing this "ping-pong" operation, program logic is such that the residual state of the intercomputer channel is in the input mode.

INTERCOMPUTER FORMATS

Message Format

There are two types of information contained in the message formats: control bits that indicate the type of format and the validity of the information, and data bits which are the actual parameters and operands being transferred (see Figure 5). Basic message formats used are:

a) Two-word—This format provides an efficient control-bit/data-bit compromise and is used for most types of transfer; and

b) Multi-word—This format allows for any data configuration that will not adapt to the two-word format.

Both formats have a common control bit area in the lower 15 bits of the first word. These bits are used to: 1) specify exactly what action is required on the data contained in the format, and 2) detect bit errors in the intercomputer transfer. Note that if the format is specified as a two-word format, the next control word can be located easily without any further decoding. If, however, the format is a multi-word format, the next control word must be located by knowing the number of data words (found in the upper 15 bits of the first word) contained in the format. Use of the entire control word reduces data transfer efficiency. Up to 45 bits of data can be contained

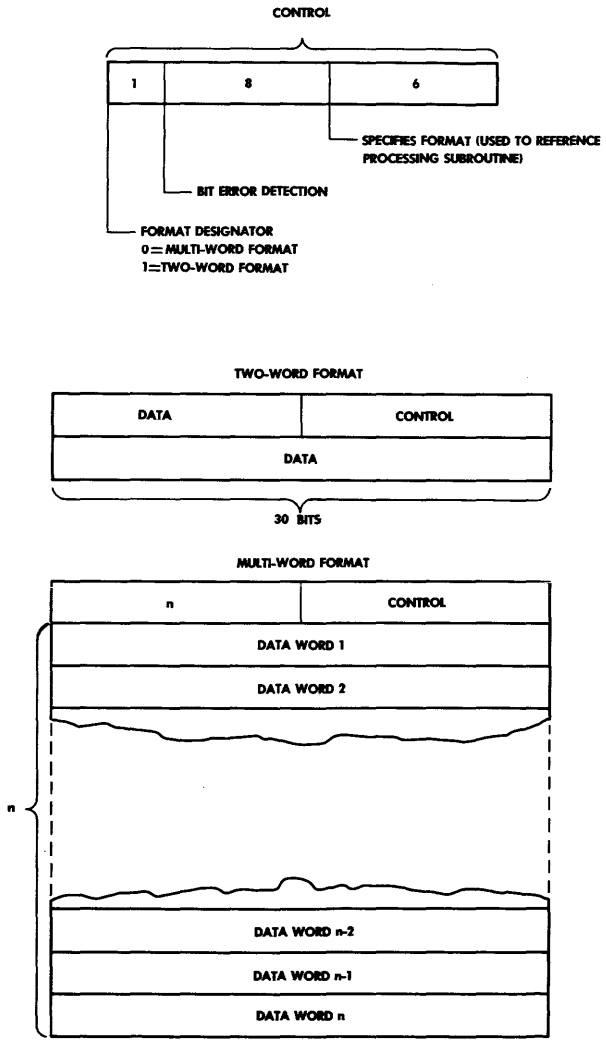


Figure 5. Intercomputer Data Formats.

in the two-word format, whereas only 30 bits of data can be contained in the first two words of a multi-word format.

Buffer Format

The message formats are superimposed on the intercomputer buffer format in chronological order. *The output buffer format's first word contains the first and last address in which the data are to be put in the receiving computer* (see Figure 6); next follow all the message formats. The last word is a unique end code to help determine whether errors have been encountered in control of the data transfer. This buffer format is thus variable in length, providing an efficiency in transmission time. Two buffer areas are provided in each computer for each transmission and one buffer area is provided for each reception. Thus, when an output buffer is active out of one area, the program may be packing the next buffer. In the case of "receive," only one buffer area is required since (as will be discussed later) all processing of a previous "receive" buffer is completed before another receive buffer will be initiated.

BUFFER CONTROL

A major control problem encountered by the program is synchronization of the buffers in the communicating computers. This control problem arises: 1) when the intercomputer data buffers are initiated, and 2) when the buffers terminate. Between initiation and termination, the buffers are maintained in synchronization by the data control signals. The method of initiating the data transfer is the main key to establishing synchronization between com-

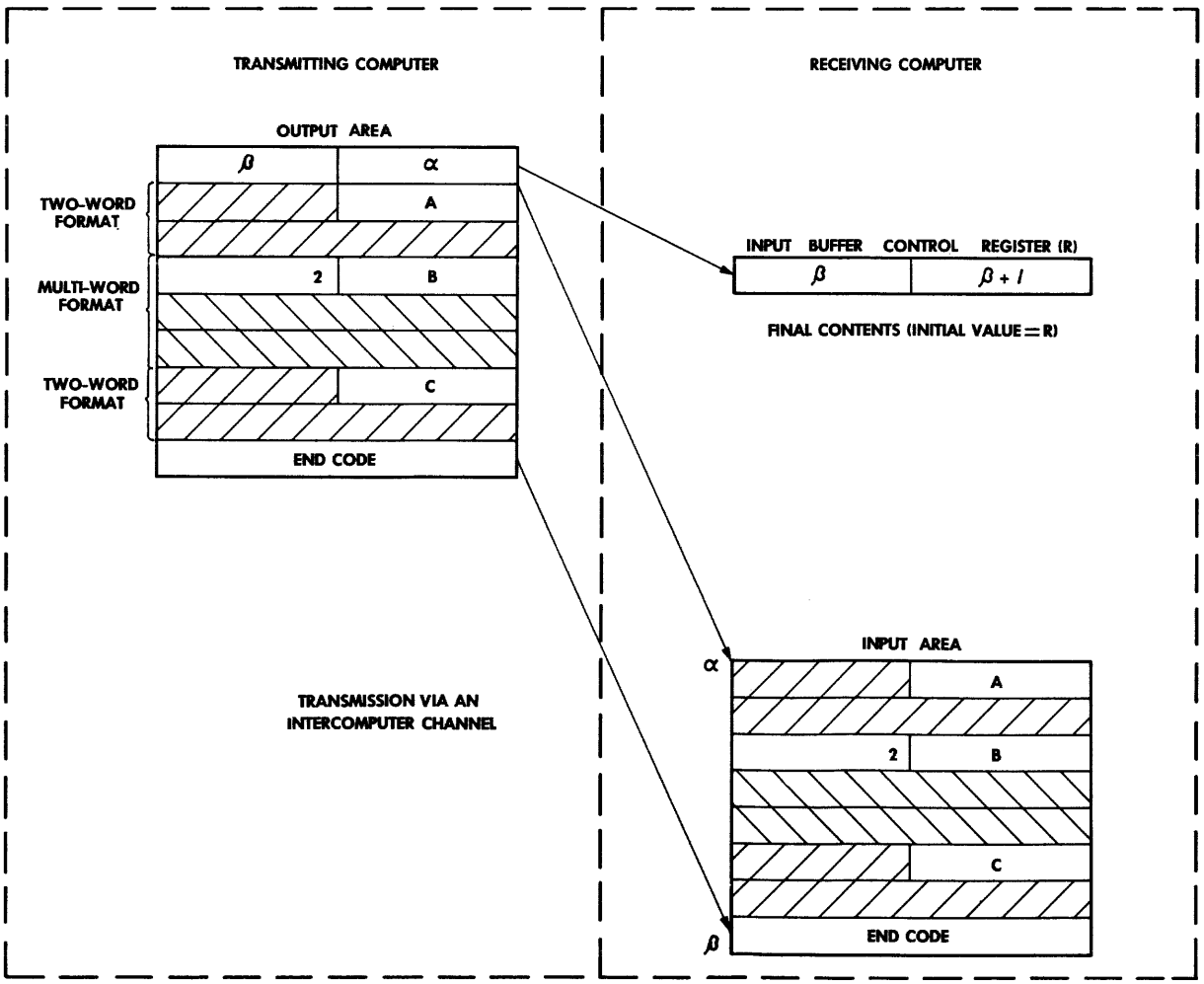


Figure 6. Intercomputer Buffer Format.

puters, because the buffer termination characteristics are defined within this initiation process.

The use of the Input Buffer Internal Interrupt by the receiving computer is predicated upon the receiving computer's knowledge of the length of the input buffer. The input buffer length must agree with the length of the output buffer from the transmitting computer. Therefore, with variable length buffers, *the transmitting computer must define the input buffer length for the receiving computer.*

The input buffer of the receiving computer is programmed to enter the first buffer word received into its Input Buffer Control Register. This is accomplished by executing an *Initiate Input Buffer (With Monitor)* instruction for the proper channel into the Input Buffer Control Register for the same channel. This instruction sets the initial address (the lower order 15 bits of the Input Buffer Control Register) the same as the address of the Input Buffer Control Register. The terminal address (the upper order 15 bits of the Input Buffer Control Register) must be different than the initial address for this method to work. Before the transmitting computer can initiate the output buffer, the receiving computer must have established an input buffer mode. The transmitting computer must determine this by testing the state of the Input Buffer Status Line with the *External Function* instruction.

When the receiving computer is ready to receive data (i.e., when all the specified conditions are met), the transmitting computer establishes the Input Buffer Control Word for the receiving computer. The first word of the output buffer data is the Input Buffer Control Word for the receiving computer. The lower order 15 bits define the initial address for data storage and the upper order 15 bits define the terminating address for data storage. Thus, the buffer area is defined for the receiving computer.

The output buffer mode is initiated by execution of the *Initiate Output Buffer (With Monitor) instruction.* As soon as this instruction is executed the first word of the output buffer data (the Input Buffer Control Word) is transmitted to the receiving computer. Data transfer proceeds under control of the Ready and Resume signals for the transmitting computer, and the Request and Acknowledge signals for the receiving computer. As soon as the transfer is completed each computer will be interrupted (as the result of the buffer monitor), allowing each computer to assume the opposite role—transmit or receive.

PROGRAM LOGIC

The intercomputer program logic is sufficiently flexible to allow operating any combination of multicomputer configurations. Three multicomputer configurations that have already been successfully operated are:

1) A three-computer system,

2) A two-computer system connected by one pair of cables, and

3) A two-computer system connected by two pairs of cables.

Refer to Figure 7 for illustration of these various configurations.

Other possible configurations for a three-computer system (connecting computers B and C) and systems utilizing four or more com-

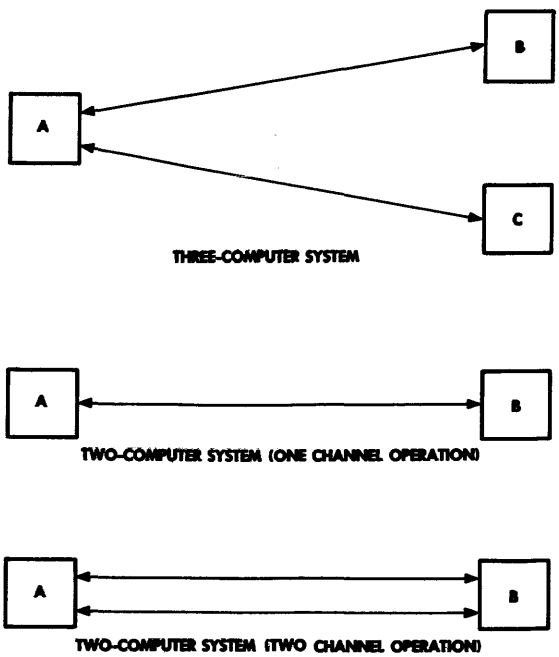


Figure 7. Multicomputer Complex.

puters could use the same program logic. A current restriction in the multicomputer system discussed here is that only two computers can be connected to any one computer.*

The discussion of program logic which follows will consider the simplest case—that of a two-computer system interconnected by one channel. Variations required for other configurations will be mentioned in the course of discussions.

Two subprograms are required for control in the two-computer configuration. These are A-B Intercomputer Control Subprogram (ABCON)†, locate in the A Computer and the B-A Intercomputer Control Subprogram, (BACON), located in the B Computer.

In addition to the subprogram in each computer, there are three Internal Interrupt routines in each computer. They are the Input Buffer Monitor, Output Buffer Monitor, and Intercomputer Failure Interrupt routines. The routines are named by suffixing the subprogram name with MIN, MOUT, and FAIL, respectively. For example, the set of interrupt routines in the A Computer for the ABCON subprogram is ABCONMIN, ABCONMOUT, and ABCONFAIL (see Figure 8). In general, one subprogram and the three associated Internal Interrupt routines are required by each computer for each channel of intercomputer operation. In the three-computer configuration shown in Figure 7, Computer A would contain two subprograms (ABCON and ACCON) whereas Computers B and C would each contain one (BACON and CACON, respectively). In the two-computer system utilizing two-channel operation, each computer would contain two subprograms.

The program logic for the subprogram and three interrupt routines is now discussed for the A-to-B data transfer. Other subprograms and Interrupt routines are identical except as noted.

*Since this paper was written, three-computer programs have been delivered connecting all computers to each other.

† The first letter in the code name indicates the computer in which the subprogram is located and the second letter indicates with which other computer the subprogram communicates.

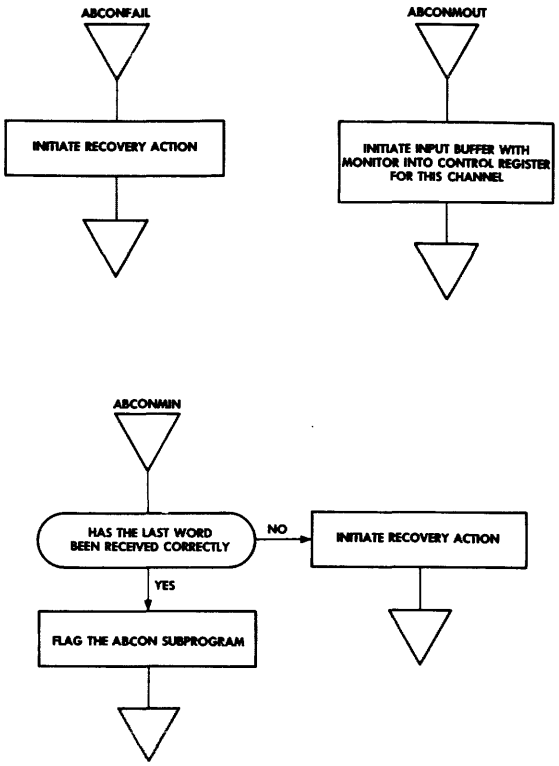


Figure 8. Interrupt Flow Diagram.

The ABCONMIN program is initiated by termination of the intercomputer input buffer. In order to terminate the buffer the first word received must contain the proper input buffer limits as computed by the BACON subprogram. The ABCONMIN program checks the last word received (designated by the input buffer control register) to make sure it is a valid end code, thereby preventing an error in the control signals from causing erroneous data to be passed between computers. If the end code is correct, this program sets the ABCON flag in the Executive Flag Table.

The Output Buffer Monitor Interrupt in the A Computer indicates that data exchange from A Computer to B Computer is complete. The A-B Output Monitor routines (ABCONMOUT) thus entered initiates an input buffer (with monitor), into the buffer control register assuring that the residual state of the intercomputer channel is in the input mode, so that an output of the B Computer is accepted immediately when initiated.

The Intercomputer Failure Interrupt routine (ABCONFAIL) is initiated when the data transfer to the B Computer has stopped for at least 32 seconds, but not more than 64 seconds (48-second average). This could occur if an input request to the B Computer were missed. The ABCONFAIL program alerts the A Computer program that a malfunction has occurred, allowing either automatic or manual remedial action to be performed, depending on the multicomputer configuration involved. For example, if both intercomputer channels are connected between the same two computers, entire data transfer is switched to the alternate channel. In any case, however, the proper information is given to allow rapid emergency maintenance whenever the faulty component is isolated within the system.

The A-B Intercomputer Control subprogram (ABCON) establishes control for the exchange of data between the A and B Computers in a multicomputer installation. This subprogram (ABCON) is flagged initially by the Input Buffer Monitor program (ABCONMIN) as described above. It is also reflagged by the EXEC on a 1-millisecond basis until the buffer is completely processed. The ABCON subprogram is referenced directly by the Executive routine in the A Computer. Operation of this subprogram is independent of other intercomputer subprograms. For example, ACCON would be contained in the A Computer in a three-computer complex.

The content of the input buffer contains both control and data information. Only the control data are processed by the ABCON subprogram. The first word of each intercomputer message format contains the control information.

The ABCON subprogram performs two basic functions: 1) processing of the data in the input buffer, and 2) control of the output buffer to the B Computer. The ABCON subprogram initiates output buffer control if all data have been processed from the input buffer.

If all data have not been processed, the ABCON program (see Figure 9) checks to determine whether the control word is valid. If the control word is not valid an error has occurred in the data bits on that intercomputer channel.

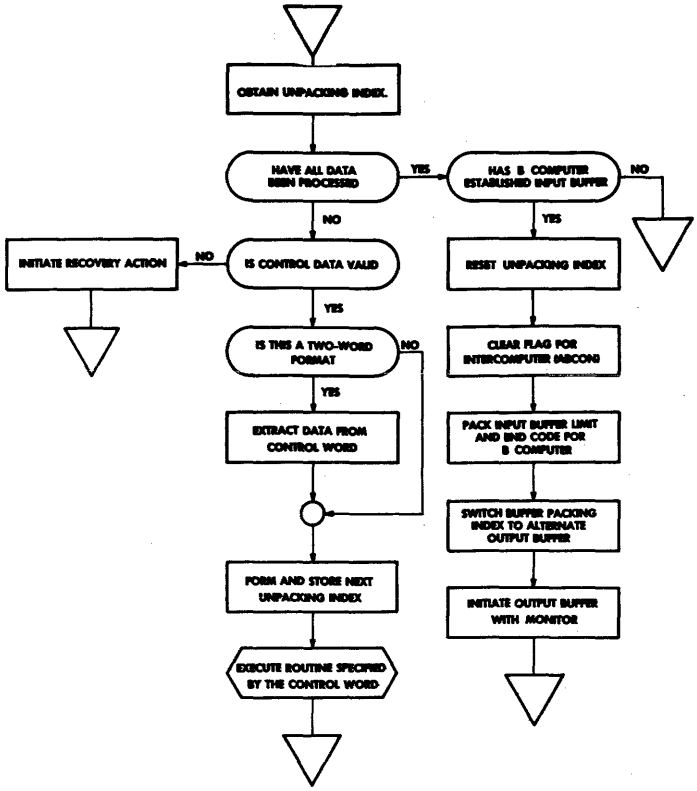


Figure 9. A-B Intercomputer Control Subprogram (ABCON).

The recovery action, as described in the ABCONFAIL routine, can be used with the difference that the discovery of error has been made in the receiving rather than in the transmitting computer.

The next operation determines the data format type so that the data can be extracted from the control word (only for two-word format). Then the index to the next control word is computed. This index is stored to make the next control word available when processing the next format. The ABCON subprogram then turns control over to the processing subroutine which is specified by the format designator in the control word. This is one application of the indirect addressing feature of the Unit Computer. The processing subroutine executes its designated task and returns control to ABCON. ABCON returns control temporarily to the Executive routine to determine whether any higher priority subprograms are scheduled.

This is similar to flag setting mechanism number 4) described in Section II of this paper.

If stringent timing requirements for higher priority tasks do not exist, all intercomputer messages could be processed before returning control to the Executive routine.

This cycle continues until all data formats have been processed. At this point the B Computer Input Buffer status is tested to determine whether the B Computer has established an input buffer. If not, control is again returned temporarily to the Executive routine. Normally this test is passed without delay and requirements for initiating an output buffer to the B Computer are begun. The unpacking index is reset for the next input buffer and the ABCON flag is cleared in the Executive Flag Table. The input buffer control word is packed in the first word of the current output buffer and the end code is packed as the last word. The output buffer control word is formed, the output buffer packing index is switched to the alternate buffer area, and an output buffer with monitor is initiated. Control is now returned to the EXEC until the next input buffer is received.

ILLUSTRATION OF OPERATION

In the discussion that follows, the method of control of the data exchange between the A and B Computers is described. The same process is used for controlling the A-C intercomputer data exchange in a three-computer system.

The computers are programmed to have equal control capabilities, but at any one time the computer that has initiated the Intercomputer Control subprogram is in control. The Intercomputer Control subprogram will retain control until all previously received inputs have been processed. When all inputs are processed, the computer in control initiates an output to the other computer. Control is relinquished to the other computer as soon as all the output data have been transferred. This type of reciprocating control is described as a "ping-pong" control system.

At the outset of system operation, the A Computer is selected to have initial control. The A Computer is forced into the output mode by an initialization subroutine. Conversely, the B Computer is started in the input mode by its initialization subroutine. Data, transmitted from the A Computer to the B Computer in the first buffer exchange, would include the A Computer's real-time clock reading (in order to provide a common time base for the entire system) in addition to other initializing data.

The ABCON subprogram in Computer A tests the special input buffer status control line that indicates when the input mode has been selected in Computer B. Computer B, via the BACONMOUT program or the initialization subroutine, must have its Input Buffer With Monitor in such a manner that the first word received from Computer A automatically defines the area of Computer B's memory in which the succeeding data words are to be placed. When the input mode is detected, the ABCON subprogram in Computer A initiates an Output Buffer With Monitor. The first word always contains the first and last address of the input area in Computer B. Data transfer proceeds under control of the regulating signals which, in the buffer mode, operate independently of the program operation. The monitor logic in both computers compare, the current data word's transfer address with the last address that should be transferred. When equal, each computer's monitor logic interrupts normal program operation to allow immediate action by the respective computers.

Computer A, which was transmitting data, initiates the Output Buffer Monitor program (ABCONMOUT), which initiates an Input Buffer With Monitor into its buffer control register, and will remain inactive in the intercomputer transfer until Computer B initiates a transfer back to Computer A. In Computer B the monitor initiates the Input Buffer Monitor program (BACONMIN), which in turn initiates the BACON subprogram, thereby accepting the intercomputer control function from Computer A. The BACON subprogram will initiate the processing of all data, using the Control bits found in the first word of each format, to select the proper processing subroutine that is required. When all data are processed, an Output Buffer With Monitor is initiated to Computer A. This buffer consists of all output data accumulated while Computer B was in the input mode.

When this buffer from Computer B to Computer A is completed, the BACONMOUT program in Computer B initiates an input buffer to accept the next output buffer from Computer A. At the same time, the ABCONMIN program in Computer A flags the ABCON subprogram so that the inputs can be processed. When the inputs have been processed by ABCON, a complete round-trip intercomputer cycle is completed. The process repeats continuously, lending to the "ping-pong" effect in data transfer.

IV. CONCLUSIONS

Other systems, both commercial and military, have been and are being conceived which require more than one computer to solve the user's problem. Some of these systems will connect two or more identical computers whereas other systems will connect a large scale central processor with smaller satellite computers. Consideration of multicomputer concepts in an original system design would allow for expansion of a single computer installation into a multicomputer installation as the problems to be solved increase or become more complex. Similar considerations would allow for greater flexibility in a multicomputer installation such that, depending on the problem or operational requirements, computers could operate separately on different problems or together to solve a common problem. The multicomputer programming techniques presented in this paper are usable as described or are adaptable to fit a particular system requirements.

The Unit Computer was designed specifically to facilitate extremely efficient intercomputer data transfer. However, such features as internal interrupts upon completion of a buffer transmission, although desirable, are not absolute requirements. The basic requirement is the ability to communicate directly with another computer. Since control signals associated with input and output to another computer differ from those associated with input/output to a peripheral device, these differences must be compensated for in the computer hardware.

The computer characteristic necessary to implement the real-time executive control philosophy is a real-time clock. This would be an internally stored, automatically maintained clock as in the Unit Computer, or an external clock capable of being referenced by the computer. The internal clock, repeat and compare instructions and external and internal interrupts included in the Unit Computer enable extremely efficient executive control; however, of these, only a real-time clock is an absolute requirement in the application of the Executive Control Philosophy to the control of a real-time system. If real-time is not a system requirement the real-time clock could be replaced by some other indexing mechanism under control of the program, thereby enabling use of the same priority logic to optimize executive control.

The Executive Control Philosophy presented in this paper is not restricted to multicomputer applications. The concepts are equally valid for a real-time system employing only one computer.

Several extensions of the techniques discussed will, no doubt, be obvious to the reader. Among these are dynamic changes in system task priorities as the operational requirements change during program operation. This merely means altering the order of the task list in the Executive stores. Included in this extension would be the ability to delete some tasks and add new tasks (subprograms) from a supplementary storage. This could be useful not only in a real-time command and control system when the nature of the battle may have changed but also in the scheduling of multiprogramming type problems in a commercial computing center. The fact that subprograms are designed to be independent of one another with any communication or control handled by means of the Executive routine and associated tables, makes the concept entirely feasible.

From a hardware standpoint, one modification considered is the ability of one computer to interrupt another computer. This characteristic has been incorporated in the later models of the Unit Computer. It has the advantage of allowing a computer to signal another computer that a buffer transmission is about to be initiated. This essentially eliminates the need for the Input Buffer Status signal and allows for more positive control of intercomputer data transfer.

ON THE ANALYSIS AND SYNTHESIS OF THREE-VALUED DIGITAL SYSTEMS

Jorge Santos and Hector Arango
Departamento de Electrotecnia
Universidad Nacional del Sur
Bahía Blanca, República Argentina

1. DEFINITIONS, SYMBOLS AND NOMENCLATURE

1.1 *Post Algebras*

Many papers about Post algebras may be found in the literature. In particular, References [1] to [4] cover the theory here required.

We shall designate the Postian variables with lower case letters a, b, etc. While most of the results next given are valid for N-valued Post algebras, we shall consider in regard to practical applications a three-valued Post algebra, that is to say, Postian variables are permitted to take values on a three elements linear lattice (Ref. [5]) (Fig. 1). Postian functions of m variables are conceived as the elements of the m-dimensional hypercube generated by m of such lattices (Ref. [6]).

Figure 1. The Three-Elements Linear Lattice.

1.2 *Operations*

The entire algebra may be generated by using the following two Primitive operations:

Cycling a'

Logical Product ab

$$\bigcap_{k=o}^{n} a_k = a_0 a_1 \ldots a_n$$

Their truth-tables are:

TABLE 1

a	a′
0	1
1	2
2	0

a \ b	0	1	2
0	0	0	0
1	0	1	1
2	0	1	2

Other operations to be used are:

Complementation $\overline{a} = ((aa')''(1a')')'$

where a″ means $(a')'$

Logical Sum $\overline{\overline{a}+\overline{b}} = (a\,b)$

$$\bigcup_{k=o}^{n} a_k = a_0 + a_1 + \ldots + a_n$$

"J" Operations $J_0(a) = (aa'')''a''$

$J_1(a) = (a'a'')''a'$

$J_2(a) = (aa')''a$

"$\overline{J}$" Operations $J_k(a) = \overline{(J_k(a))}$ $\quad k \in \{0,1,2\}$

"K" Operations $K(a) = 1\,\overline{J}_2(a)$

Their truth-tables are easily obtained from their definitions in terms of the primitive operations.

$$f(x_1, x_2, \ldots, x_n) =$$
$$= \cup f(k_1, k_2, \ldots, k_n)\ J_k\ (x_1)\ J_k\ (x_2) \ldots J_k\ (x_n)$$
$$= \cap f(k_1, k_2, \ldots, k_n) + \overline{J_k}\ (x_1) + \overline{J_k}\ (x_2) + \ldots + \overline{J_k}\ (x_n)$$

where $k_1, k_2, \ldots, k_n \in \{0,1,2\}$, and therefore the symbols $\cup$ and $\cap$ stand for 3^n terms.

For the sake of conciseness we shall use the notation

$$(k_1 k_2 \ldots k_n) = J_k\ (x_1)\ J_k\ (x_2) \ldots J_k\ (x_n)$$

The symbol $\emptyset$ instead of a k in the parentheses will imply that the corresponding J-operation in the second member must be omitted.

1.4 *Basic Logical Units Symbols*

In the following, logical diagrams will be often used. They are formed by interconnecting basic units, each one performing some definite logical operation. The symbols adopted for such units are listed in Fig. 2.

2. ELECTRICAL MODELS FOR POST ALGEBRAS

The logical values 0,1,2 will be associated to voltage levels according to Table 2.

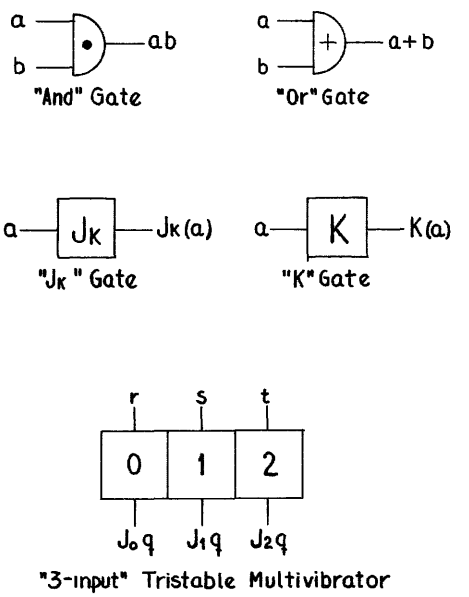


Figure 2. Symbols Used for the Basic Logical Units.

1.3 *Canonical Forms*

From the lattice concept of Postian functions described in 1.1 the following canonical forms are obvious:

TABLE 2

Logical value	Associated voltage level (in volts)
0	+6
1	0
2	—6

Given a certain Postian function $f(x_1, x_2, \ldots, x_n)$, a circuit with n inputs $(V_1, V_2, \ldots, V_n)$ and one output V is said to be a Model of the function, if for any given combination of voltage levels at the input, the output assumes a voltage such that its relation to the set of inputs is isomorphic with the relation between the function and its independent variables.

As usual in the Synchronous type of Computers, those voltage levels are analyzed at certain regular intervals of time (clock pulses).

Postian variables may be related to other physical magnitudes of a system, such as currents, magnetization states, etc., provided that they appear in three discriminable values. Those systems are also models of Postian functions in the general sense stated above. The problem of finding the Postian functions associated with a given system is usually called "Logical analysis."

In Table 3, adopted correspondences between Postian variables and physical magnitudes are given:

TABLE 3

Logical value	Current	Magnetization
0	Positive sign	Positive sign
1	No current	Film demagnetized
2	Negative sign	Negative sign

3. LOGICAL ANALYSIS

3.1 *Introduction*

In the consideration of such *ternary* systems, we must distinguish between *combinational* and *sequential systems*. The meaning of such terms will be the usual one.[3,7] The analysis of both kinds of ternary systems is already known and may be easily performed. The correspondence between physical and logical values is first established. Next, the dependence of the output electrical values on the input electrical values is obtained (to carry out this step a definite knowledge about the physical behavior of the system is needed). The truth-table follows easily, and from it the canonical form of the Postian functions can be obtained.

3.2 *Application to the Analysis of a Ternary Store*

To illustrate the point, we shall consider the logical analysis of a magnetic thin-film ternary store. The design is based on the technique for magnetic thin-film binary storage described in Ref. [8].

A Ni-Fe alloy is deposited on a cylindrical glass tube by evaporation in the presence of a magnetic field. The resulting film exhibits a considerable anisotropy: an "easy" (rectangular loop) direction of magnetization is established in the film parallel to the applied magnetic field, and a "hard" (linear characteristic) direction perpendicular to the previous one.

In the following, we shall consider as relevant magnitudes: 1) Input currents, 2) Remanent magnetization states, 3) Voltage outputs.

In Fig. 3 an isolated specimen of a thin-magnetic film is represented.

There are two exciting wires, X, Y. Current flowing in each wire produces a field in the corresponding coordinate. The "easy" direction is circumferential, so remanent magnetic states can only occur in the X-direction.

For suitable values of X and Y pulses, the following results were obtained:

1) An X-pulse does not change significantly any previous saturation state of the film.
2) A "nondestructive" intensity Y-pulse does not change significantly any previous saturation or demagnetized state. But if the pulse is increased to "destructive" intensity, the film always becomes damagnetized.
3) An X-pulse in coincidence with a nondestructive Y-pulse produces in every case a remanent magnetization state in a circumferential direction, its sign depending on the X-pulse polarity.

Word selection and nondestructive reading

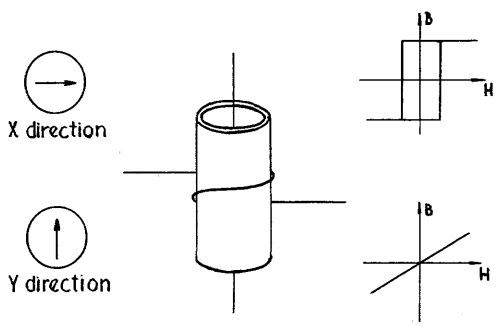


Figure 3. An Isolated Specimen of Magnetic Thin Film, showing its magnetic characteristic on the X and Y Directions.

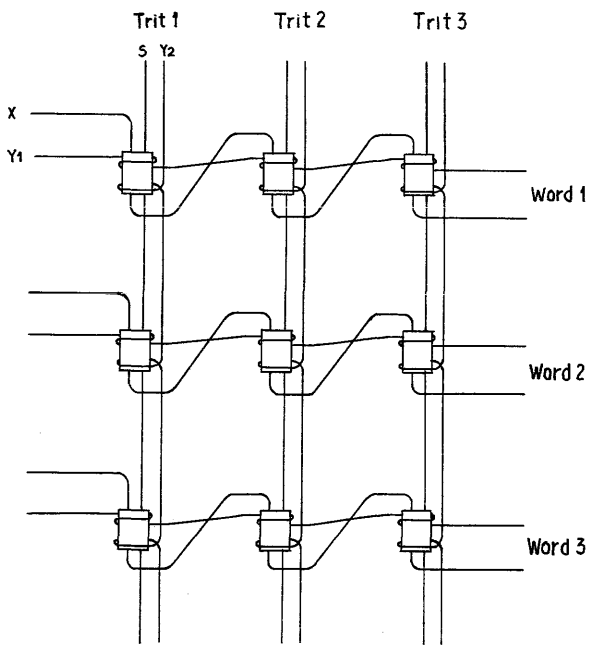


Figure 4. Word Selection Array. Three Words of Three Bits Each.

would be therefore achieved with the array illustrated in Fig. 4. Every ring can store a "trit" (Log_2 3 bits) of information. Noncoincident Y-pulses are made nondestructive, whereas coincident Y-pulses generate a resulting MMF leading to ring demagnetization.

The process of writing should be performed as follows: By the X, Y_1 wires of the selected word, current pulses are sent as indicated in Fig. 5. The writing of a 0, 1 or 2 depends on the temporal position of the aiding pulse Y_2, as can be easily understood from inspection of the same figure. Note that the information is replaced only in the films of the selected word.

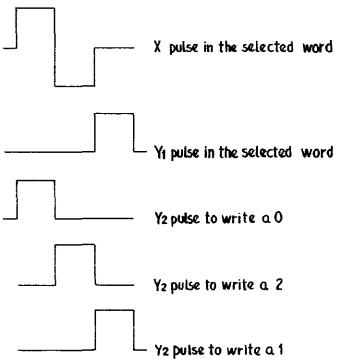


Figure 5. Writing Pulses in the Ternary Store.

Reading is performed by sending a pulse along the Y_1-wire of the selected word. If a 0 or a 2 are stored, responses would be obtained as illustrated in Fig. 6. If the specimen was demagnetized—a 1 stored—the readout wire would not detect any change of flux and no response would appear.

We shall use the following Postian variables:

TABLE 4

Postian variables	Associated physical magnitudes
x, y_1, y_2	Currents in the corresponding input wires.
m	Remanent magnetization state of the thin-film.
s	Normalized voltage output from readout wire S.

Then we may construct the truth-table of the store (Table 5).

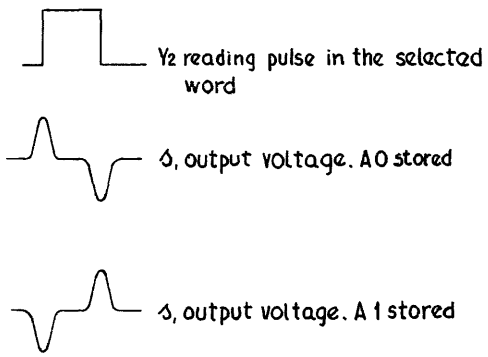


Figure 6. Reading Pulse and Outputs in the Ternary Store.

TABLE 5

Operation	x	y_1	y_2	m^n	m^{n+1}	s
Write 0	0	1	0	0	0	0
	0	1	0	1	0	0
	0	1	0	2	0	0
	0	1	1	0	0	1
	0	1	1	1	0	2
	0	1	1	2	2	2
Write 1	1	0	0	0	1	0
	1	0	0	1	1	1
	1	0	0	2	1	2
Read	1	0	1	0	0	0
	1	0	1	1	1	1
	1	0	1	2	2	2
	1	1	0	0	0	0
	1	1	0	1	1	1
	1	1	0	2	2	2
	1	1	1	0	0	1
	1	1	1	1	1	1
	1	1	1	2	2	1
Write 2	2	1	0	0	2	2
	2	1	0	1	2	2
	2	1	0	2	2	2
	2	1	1	0	0	0
	2	1	1	1	2	0
	2	1	1	2	2	1

Other combinations of X, Y_1 and Y_2 pulses do not occur in the rings, and therefore they may be considered as *forbidden* or *don't care* terms. From the truth-table, the canonical forms of m^{n+1} and s follows immediately.

In 6.3, a method for transferring the stored information to a MVs register shall be developed.

4. MINIMIZATION OF POSTIAN FUNCTIONS

Minimization can be performed by an extension of the Quine method currently used for Boolean functions. The Absorption law may be here expressed as

$$\bigcup_{k \in \{0,1,2\}} J_k(x)\, f = f$$

where f stands for any Postian function.

In this way it is possible to obtain a set of prime implicants. Tabulating them against the function minterms, we may determine the essential terms, and then choose between the nonessential terms a set of prime implicants covering the function minterms not included in the set of essential terms.

Redundancies (forbidden or don't care terms) are treated as in the Boolean case, and a minimal form of the Postian function is obtained. From this simplest form it is an easy matter to draw the corresponding logical diagram.

As an example, we shall proceed to minimize the canonical form of m^{n+1} derived from truth-table 5.

First of all, the 3^4 four-variable minterms are grouped as in Table 6 a,b.

TABLE 6 (a)

$m^{n+1} = 0$	$m^{n+1} = 1$	$m^{n+1} = 2$
0100	1000	0112
0101	1001	1012
0102	1002	1102
0110	1011	1112
0111	1101	2100
1010	1111	2101
1100		2102
1110		2111
2110		2112

TABLE 6 (b)

F B		
0000	1020	2000
0001	1021	2001
0002	1022	2002
0010	1120	2010
0011	1121	2011
0012	1122	2012
0020	1200	2020
0021	1201	2021
0022	1202	2022
0120	1210	2120
0121	1211	2121
0122	1212	2122
0200	1220	2200
0201	1221	2201
0202	1222	2202
0210		2210
0211		2211
0212		2212
0220		2220
0221		2221
0222		2222

Note that for the combinations of x, y_1, y_2, m^n values indicated by the minterms of group $m^{n+1} = 0$ the functions take the value 0. In the same way, for groups $m^{n+1} = 1$ and 2, the functions take the values 1 and 2 respectively. Finally, the group F B is composed by all forbidden combinations.

Next, the following steps are performed:

1) Each minterm of group $m^{n+1} = 1$ is checked with every other in the same group or in group F B in order to apply the Absorption law.

2) Each minterm of group $m^{n+1} = 2$ is checked with every other in the same group or in group F B in order to apply the Absorption law.

3) With a similar purpose, each minterm of group F B is checked with every other of the same group.

In Table 7 are listed all the functions generated by one application of the Absorption law to all the four variable minterms.

Those functions arising from steps 1), 2) or 3) are distinguished by adding a 1, 2 or f respectively.

TABLE 7

000Ø	f	100Ø	1	200Ø	f
00Ø0	f	10Ø0		20Ø0	f
0Ø00		1Ø00		2Ø00	2
Ø000	1	Ø100		Ø200	f
001Ø	f	101Ø		201Ø	f
00Ø1	f	10Ø1	1	20Ø1	f
0Ø01		1Ø01	1	2Ø01	2
Ø001	1	Ø101		Ø201	f
002Ø	f	102Ø	f	202Ø	f
00Ø2	f	10Ø2		20Ø2	f
0Ø02		1Ø02		2Ø02	2
Ø002	1	Ø102		Ø202	f
010Ø		110Ø		210Ø	2
01Ø0		11Ø0		21Ø0	
0Ø10		1Ø10		2Ø10	
Ø010		Ø110		Ø210	f
011Ø		111Ø		211Ø	
01Ø1		11Ø1	1	21Ø1	2
0Ø11		1Ø11	1	2Ø11	2
Ø011	1	Ø111		Ø211	f
012Ø	f	112Ø	f	212Ø	f
01Ø2		11Ø2	2	21Ø2	2
0Ø12	2	1Ø12	2	2Ø12	2
Ø012	2	Ø112	2	Ø212	f
020Ø	f	120Ø	f	220Ø	f
02Ø0	f	12Ø0	f	22Ø0	f
0Ø20	f	1Ø20	f	2Ø20	f
Ø020	f	Ø120	f	Ø220	f
021Ø	f	121Ø	f	221Ø	f
02Ø1	f	12Ø1	f	22Ø1	f
0Ø21	f	1Ø21	f	2Ø21	f
Ø021	f	Ø121	f	Ø221	f
022Ø	f	122Ø	f	222Ø	f
02Ø2	f	12Ø2	f	22Ø2	f
0Ø22	f	1Ø22	f	2Ø22	f
Ø022	f	Ø122	f	Ø222	f

These marked functions are checked as in 1), 2) and 3) for a new application of the Absorption law. In Table 8 are listed all the functions resulting from two applications of such law, with a similar indication for those arising from marked functions.

Of course, those functions not discarded in the first absorption process must be considered as prime implicants. Here the only prime implicant generated at this stage is (11Ø2).

TABLE 8

00Ø0	f	10Ø0		20Ø0	
0Ø0Ø		1Ø0Ø		2Ø0Ø	2
Ø00Ø	1	Ø10Ø		Ø20Ø	f
0Ø00		1Ø00		2Ø00	
Ø0Ø0		Ø1Ø0		Ø2Ø0	f
ØØ00		ØØ10		ØØ20	f
01Ø0		11Ø0		21Ø0	
0Ø1Ø		1Ø1Ø		2Ø1Ø	
Ø01Ø		Ø10Ø		Ø21Ø	f
0Ø01		1Ø01	1	2Ø01	2
Ø0Ø1	1	Ø1Ø1		Ø2Ø1	f
ØØ01		ØØ11		ØØ21	f
02Ø0	f	12Ø0	f	22Ø0	
0Ø2Ø	f	1Ø2Ø	f	2Ø2Ø	
Ø02Ø	f	Ø12Ø	f	Ø22Ø	f
0Ø02		1Ø02		2Ø02	2
Ø0Ø2		Ø1Ø2		Ø2Ø2	f
ØØ02		ØØ12	2	ØØ22	f

An exploration of the new marked functions for a third application of the Absorption law does not give us any result. Therefore, all the functions of Table 8 marked 1 or 2 are prime implicants of the original function.

Finally, in Table 9 a, b the prime implicants are tabulated against the corresponding function minterms in order to get subgroups of prime implicants covering all the minterms in each case.

TABLE 9

(a)

	1 0 0 0	1 0 0 1	1 0 1 0	1 0 1 1	1 1 0 1	1 1 1 1
Ø00Ø	x	x	x			
Ø0Ø1		x		x		
1ØØ1					x	x

(b)

	0 1 1 2	1 0 1 2	1 1 0 2	1 1 1 2	2 1 0 0	2 1 1 1	2 1 0 2	2 1 1 2	2 1 2 2
ØØ12	x	x		x					x
2Ø0Ø					x	x	x		
2Ø01						x		x	
2Ø02							x		x
11Ø2			x	x					

This last step produces the following minimal form: $m^{n+1} = (11\emptyset 2) + (\emptyset\emptyset 12) + (2\emptyset 0\emptyset) + (2\emptyset\emptyset 1) + 1\,((\emptyset 00\emptyset)+(\emptyset 0\emptyset 1)+(1\emptyset\emptyset 1))$.

5. THE SYNTHESIS PROBLEM. COMBINATIONAL CIRCUITS

5.1 *Introduction*

The matter of obtaining an electrical model of a given function is currently described as synthesis. We shall consider combinational circuits first.

Of course, synthesis is not a problem with a single answer. As in binary synthesis, the first task is to design a few basic circuits, each one performing some particular operation. The set of operations implemented must be complete, in the sense that the entire algebra could be generated from them (see Ref. [9]). In order to get reliability, we shall impose an "on-off" performance of the transistors used in the circuitry. Reshaping of the signal in every active stage will be secured by using common emitter connections.

5.2 *Possible Operations*

Let us briefly discuss our possibilities. For PNP transistors we have the fundamental circuit indicated in Fig. 7. The points c, e admit the following voltage combinations (Table 10).

TABLE 10

c	e
−6	0
−6	+6
0	+6

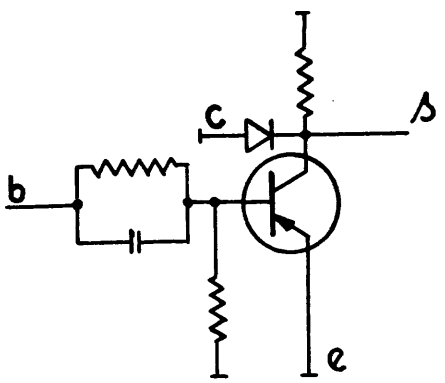


Figure 7. Fundamental Connection for the Active Stages.

The value of the bias resistance defines the behavior of the circuit as Type I or II, according to the following table (Table 11).

TABLE 11

b	Transistor	
	Type I	Type II
+6	off	off
0	off	on
−6	on	on

Therefore, the following truth-tables can be written:

TABLE 12

c	e	Type	Input	Output	Postian Function
2	1		0	2	
2	1	I	1	2	$1 + \overline{J}_2(b)$
2	1		2	1	
2	1		0	2	
2	1	II	1	1	$1 + J_0(b)$
2	1		2	1	
2	0		0	2	
2	0	I	1	2	$\overline{J}_2(b)$
2	0		2	0	
2	0		0	2	
2	0	II	1	0	$J_0(b)$
2	0		2	0	
1	0		0	1	
1	0	I	1	1	$1\,\overline{J}_2(b)$
1	0		2	0	
1	0		0	1	
1	0	II	1	0	$1\,J_0(b)$
1	0		2	0	

We shall implement "And" and "Or" gates by means of conventional diode circuits. The so called "Constant Functions" 0, 1, 2 are of course generated as their corresponding voltage levels. Two properly selected functions of Table

12, together with the previous ones, allow the generation of the entire algebra.

A convenient pair appears to be J_0 and $\overline{J}_2$, leading to simple expressions for J_1 and J_2, such as

$$J_2(x) = J_0(\overline{J}_2(x))$$
$$J_1(x) = J_0(J_0(x) + \overline{J}_2(x))$$

As we see, the system of operations (J_0, $\overline{J}_2$, , +) is certainly complete. However, because of electronic reasons that will become evident later, the operation

$$K(x) = 1\,\overline{J}_2(x)$$

will be also considered.

Any Postian function can then be systematically generated. In order to get reshaping of the signal after every degenerative stage, the complements of the minterms ($k_1k_2 \ldots k_n$) are sometimes generated, either using deMorgan's Laws or the relation

$$J_0(\text{Minterm}) = \overline{\text{Minterm}}$$

Also we shall take advantage of the relation

$$K(\overline{\text{Minterm}}) = 1\,\text{Minterm}$$

Note that $\overline{J}_0$ and $\overline{J}_1$ can be generated as

$$\overline{J}_0(x) = \overline{J}_2(J_0(x))$$
$$\overline{J}_1(x) = J_0(\overline{J}_0(x)\,\overline{J}_2(x))$$

5.3 *Example*

The function

$$J_0(x)\,J_1(y) + 1\,J_2(x)\,J_2(y)$$

is realized as in Fig. 8.

5.4 *Implementation*

Fig. 9 illustrates the electronic circuits performing the set of selected operations.

6. THE SYNTHESIS PROBLEM. SEQUENTIAL CIRCUITS

6.1 *Introduction*

The Tabular and Matrix techniques[10,11] used in sequential Boolean synthesis can be easily extended to Postian systems.

As a basic sequential unit, the electronic circuit of Fig. 10 was developed. The analysis of the circuit leads to the following truth-table, where n indicates the order number of clock pulse:

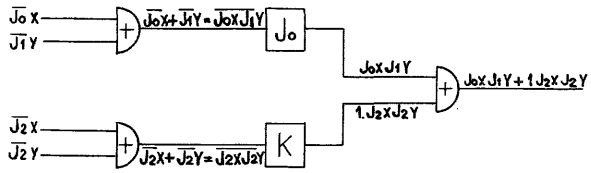


Figure 8. An Example of Realization of a Two Variables Postian Function.

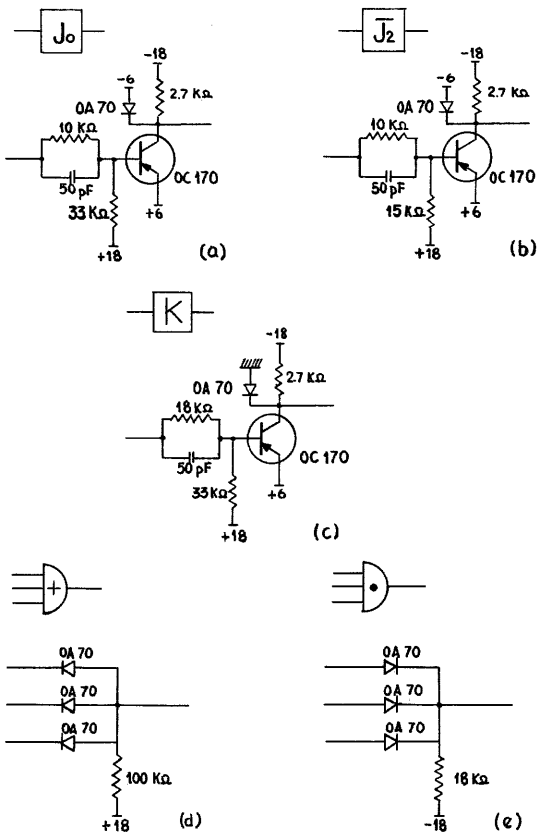


Figure 9. Electronic Diagram of the Basic Combinational Circuits Selected.

TABLE 14

r^n	s^n	t^n	q^{n+1}
0	2	2	0
2	0	2	1
2	2	0	2
2	2	2	q^n

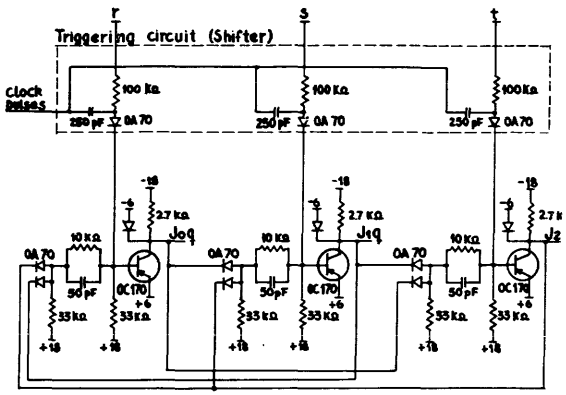


Figure 10. Electronic Diagram of a "3-input" Tristable Multi-vibrator, Selected as the Basic Sequential Unit.

All other input combinations are supposed forbidden. So we shall have, after a convenient minimization process, the following characteristic equation:

$$q^{n+1} = 1\ (\emptyset 0 \emptyset) + (\emptyset\emptyset 0) + (222)\ q^n$$

in the independent variables r^n, s^n, t^n.

For obvious reasons, this circuit will be called a "3-input" Tristable Multivibrator. It represents some sort of generalization of the well known RS flip-flop.

By combining "3-input" MVs. with the combinational units already described, any sequential Postian function can be synthetized. In the following paragraphs some examples are given on the matter.

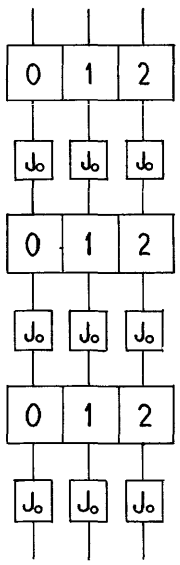


Figure 11. Logical Diagram of a Shift Register.

6.2 *Other Sequential Circuits*

By a proper connection of its inputs and outputs, a 3-input Tristable Multivibrator can be made to behave either like a "Cycling" MV or like a "Delay" MV, whose truth-tables are indicated in Table 15.

TABLE 15

(a) Cycling

i	q^n	q^{n+1}
0	0	1
0	1	2
0	2	0
1	0	0
1	1	1
1	2	2
2	0	2
2	1	0
2	2	1

(b) Delay

i	q^n	q^{n+1}
0	0	0
0	1	0
0	2	0
1	0	1
1	1	1
1	2	1
2	0	2
2	1	2
2	2	2

By applying the tabular method of synthesis, it is an easy matter to obtain the input equations for the 3-input MV:

Simulation of Cycling MV	Simulation of Delay MV
$r = \overline{(02)}\ \overline{(21)}$	$r = \overline{(0\emptyset)} = \overline{J}_0 q$
$s = \overline{(00)}\ \overline{(22)}$	$s = \overline{(1\emptyset)} = \overline{J}_1 q$
$t = \overline{(01)}\ \overline{(20)}$	$t = \overline{(2\emptyset)} = \overline{J}_2 q$

where the variables of the minterms are i, q.

Another point of interest is the design of shift registers. From a logical point of view, a shift register is merely a chain of delay MV. To take advantage of the fact that in our 3-input MV we have the three J-functions on the output, we may use the connection illustrated in Fig. 11.

The fact that any MV may be changing while influencing the next, is taken care of by the

delay introduced by the time constants of the shifter circuits. (See Fig. 10.)

6.3 *Other Examples*

Let us design a decade counter by the Tabular method. The left part of Table 16 represents the Application Equations in the form of a truth-table giving the desired evolution of the outputs of the counter multivibrators. On the right we have written suitable combinations of the inputs, taken from Table 14.

TABLE 16

q_0^n	q_1^n	q_2^n	q_0^{n+1}	q_1^{n+1}	q_2^{n+1}	r_0^n	s_0^n	t_0^n	r_1^n	s_1^n	t_1^n	r_2^n	s_2^n	t_2^n
0	0	0	0	0	1	0	2	2	2	2	2	2	0	2
0	0	1	0	0	2	0	2	2	2	2	2	2	2	0
0	0	2	0	1	0	0	2	2	2	0	2	0	2	2
0	1	0	0	1	1	0	2	2	2	2	2	2	0	2
0	1	1	0	1	2	0	2	2	2	2	2	2	2	0
0	1	2	0	2	0	0	2	2	2	2	0	0	2	2
0	2	0	0	2	1	0	2	2	2	2	2	2	0	2
0	2	1	0	2	2	0	2	2	2	2	2	2	2	0
0	2	2	1	0	0	2	0	2	0	2	2	0	2	2
1	0	0	0	0	0	0	2	2	2	2	2	2	0	2

Then the Input Equations can be easily obtained. Once minimized, they adopt the form

$r_o = (022)$ $\quad r_1 = \overline{(022)}$ $\quad r_2 = \overline{(0\emptyset 2)}$

$s_o = \overline{(022)}$ $\quad s_1 = \overline{(002)}$ $\quad s_2 = (0\emptyset 0)$

$t_o = 2$ $\quad t_1 = \overline{(012)}$ $\quad t_2 = (0\emptyset 1)$

The resulting logical diagram is illustrated in Fig. 12.

As a final application of this method, we shall develop a circuit to transfer a word of the thin film store described in 3.2 to a Register of 3-input MVs. The required truth-table may be simplified, by proper use of redundancies, to (Table 17)

TABLE 17

Operation	x	y_1	y_2	q^{n+1}
Reading	1	∅	1	m^n
Writing	0	∅	0	q^n
	2	∅	2	q^n

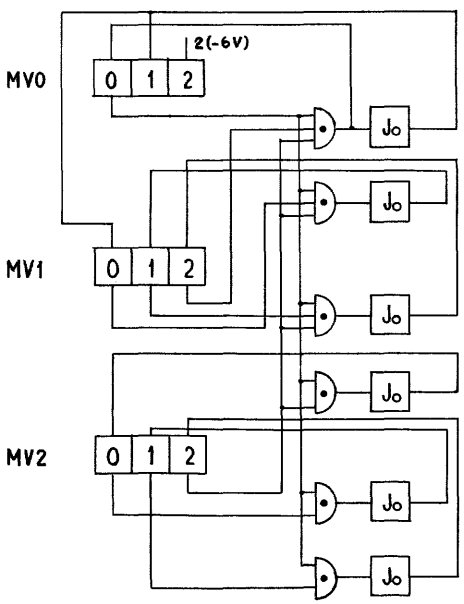


Figure 12. Logical Diagram Showing a Base Three Decade Counter.

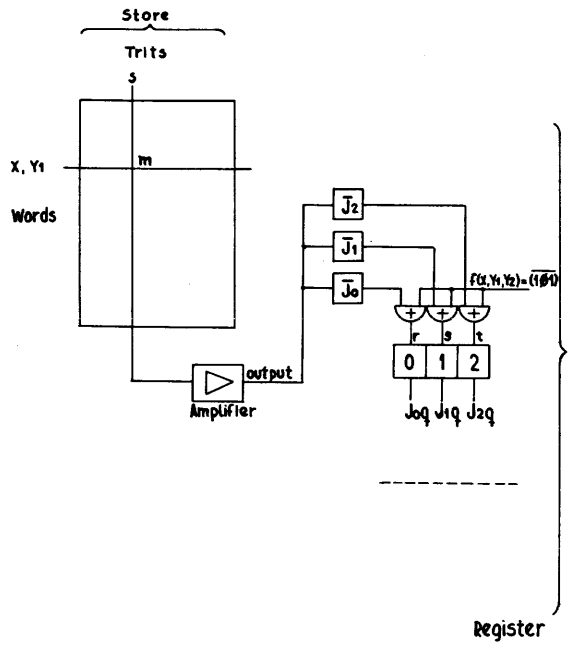


Figure 13. Concerning to the Transfer of Information from the Thin-Film Store to a MVs. Register.

That is to say, each MV copies the state of its associated ring only while reading.

After obtaining and simplifying the input equations, we shall have the logical diagram indicated in Fig. 13.

7. DESIGN OF A TERNARY FULL ADDER

Kilburn et al. have developed a fast carry-propagation binary adder,[12] now incorporated in the Arithmetic Unit of ATLAS Ferranti Computer.[13] Let us synthesize the ternary version of such system.

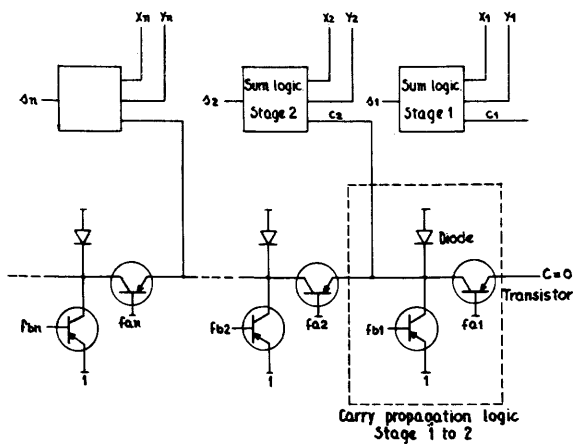
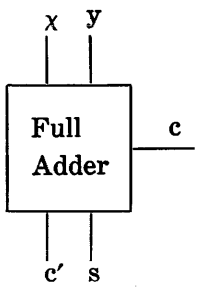


Figure 14. Block Diagram of the Kilburn Adder.

The truth-table of the ternary sum is indicated in Table 18.

TABLE 18

c	x	y	s	c′
0	0	0	0	0
0	0	1	1	0
0	0	2	2	0
0	1	0	1	0
0	1	1	2	0
0	1	2	0	1
0	2	0	2	0
0	2	1	0	1
0	2	2	1	1
1	0	0	1	0
1	0	1	2	0
1	0	2	0	1
1	1	0	2	0
1	1	1	0	1
1	1	2	1	1
1	2	0	0	1
1	2	1	1	1
1	2	2	2	1



The case $c = 2$ is not included because the propagation of any carry value other than 0 or 1 is not possible in a full adder, whichever the numerical base used.

From the table, the canonical forms of c′, s ar obtained, and after a minimization process, we have

$$c' = f_a(x,y)\, c + f_b(x,y)$$

$$s = 1\,(f_c(x,y)\, J_o(c) + f_d(x,y)\, J_1(c) + f_a(x,y)\, J_o(c) + f_c(x,y)\, J_1(c)$$

where

$$f_a(x,y) = (02) + (11) + (20)$$

$$f_b(x,y) = (12) + (21) + (22)$$

$$f_c(x,y) = (01) + (10) + (22)$$

$$f_d(x,y) = (00) + (12) + (21)$$

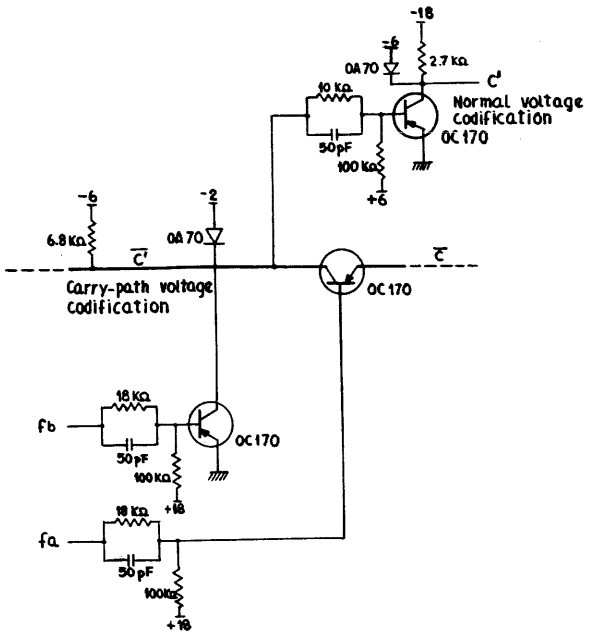


Figure 15. Carry Propagation: Electronic Diagram of a Typical Stage, Including Normalization of Voltage Levels.

With the logic illustrated in Fig. 14 the carry propagation time is reduced to the sum of the switching time of the slowest carry-path transistor and the time needed for the carry pulse to travel along the carry-path.

Because of electronic reasons, another correspondence between Postian and electrical values is adopted on the carry-path circuit, namely, 0 volt for the "1" and —2 volts for the "2". Therefore, those circuits analyzing carry-path information must be designed to match such correspondence.

The electronic diagram of a typical carry propagation stage is illustrated in Fig. 15.

The sum s is easily synthesized by the method described in 5.3, leading to the gate circuit of Fig. 16. The storage of s in a MVs. register is also indicated.

If the Arithmetic Unit is of the serial-parallel

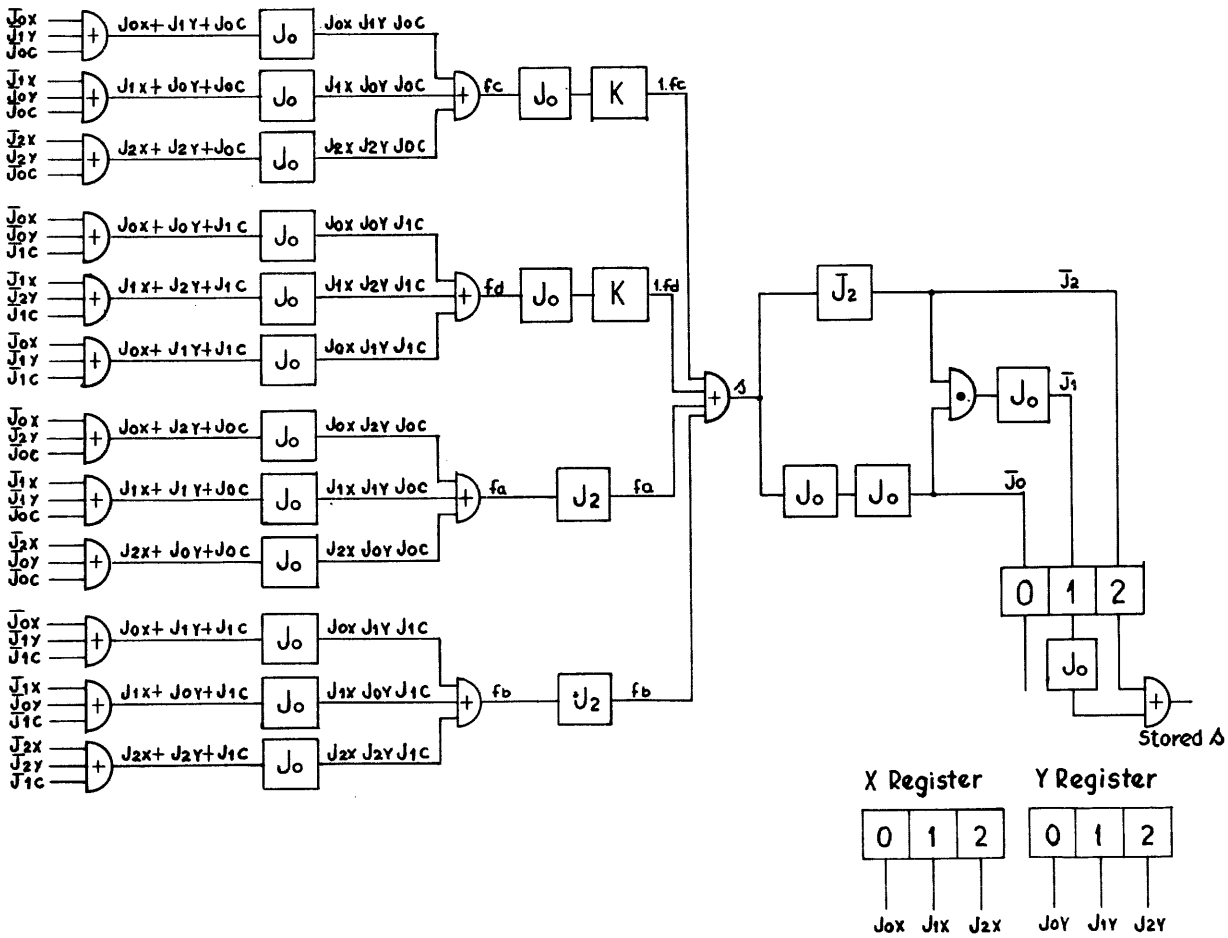


Figure 16. Generation and Storage of the Sum.

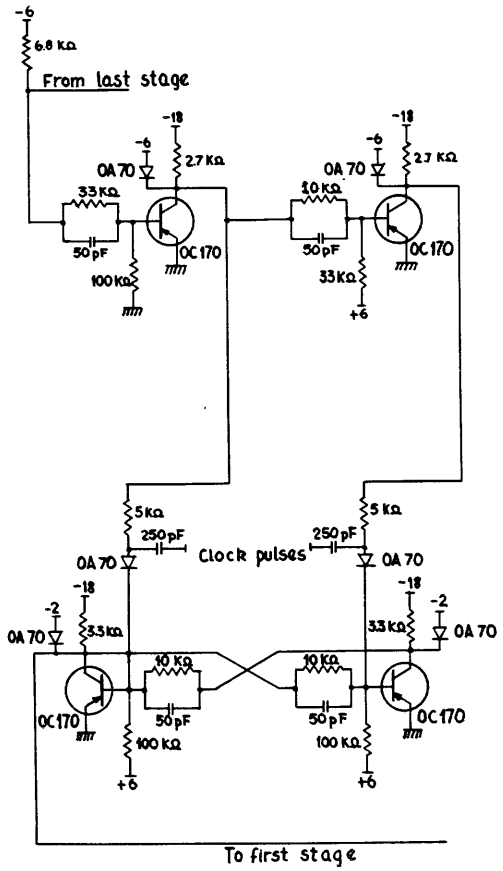


Figure 17. Carry Feedback on a Serial-Parallel Adder.

type, the carry generated at the final stage at time n must be stored in a Delay MV in order to incorporate it as input carry at the least significant digit stage at time n + 1. Because of the binary character of the carry propagation, the carry storage may be performed by means of a conventional binary Delay MV, as illustrated in Fig. 17.

ACKNOWLEDGEMENT

The authors wish to express their appreciation to Mr. F. Lorenzo who contributed to the actual construction of the Kilburn adder.

REFERENCES

1. POST, E. L. "Introduction to a general theory of elementary propositions," American Journal of Mathematics, Vol. 43, pp. 163–85, 1921.
2. ROSSER, J. B., and TURQUETTE, A. R. "Axiom schemes for M-valued propositional calculi," Journal of Symbolic Logic, Vol. 10, pp. 61–82, 1945.
3. LEE, C. Y., and CHEN, W. H. "Several-Valued Combinational switching circuits" AIEE Transactions, Vol. 75, Part I (Communication and Electronics), pp. 278–83, July, 1956.
4. VACCA, R. "A three-valued system of logic and its application to base three digital circuits," Information Processing — Proceedings of the International Conference on Information Processing, UNESCO. Paris, 15–20, June, 1959, pp. 407–14.
5. BIRKHOFF, G. "Lattice Theory," Amer. Math. Soc., New York, 1940.
6. LEE, C. Y. "Switching functions for an N-dimensional cube," AIEE Transactions, Vol. 73, Part 1, pp. 289–91, September 1954.
7. PHISTER, M. "Logical design of digital computers," John Wiley & Sons, New York, 1956.
8. HOFFMAN, J. R., TURNER, J. R., and KILBURN, T. "High-speed Digital Storage using Cylindrical Magnetic Films," Journal of the British I.R.E., Vol. 20, No. 1, pp. 31–6, January, 1960.
9. BERLIN, R. D. "Synthesis of N-valued switching circuits," IRE Transactions on Electronic Computers, Vol. EC–7, No. 1, pp. 52–7, March 1958.
10. ARANT, G. W. "A Time-sequential tabular analysis of flip-flop logical operation," IRE Transactions on Electronic Computers, Vol. EC–6, No. 2, pp. 72–4, June 1957.
11. LEDLEY, R. S. "Boolean Matrix Equations in digital circuit design," IRE Transactions on Electronic Computers, Vol. EC–8 No. 2, pp. 131–9, June 1959.
12. KILBURN, T., EDWARDS, D. B. H., and ASPINALL, D. "A parallel arithmetic unit using a saturated transistor fast-carry circuit," Proceedings of the I.E.E. (Brit.), Vol. 107, Part B, pp. 573–84, November 1960.
13. KILBURN, T., HOWARTH, D. J., PAYNE, R. D., and SUMNER, P. H. "The Manchester University ATLAS Operating System — Part I: Internal Organization," The Computer Journal, Vol. 4, No. 3, pp. 222–5, October 1961.

AN ALGORITHM FOR PLACEMENT OF INTERCONNECTED ELEMENTS BASED ON MINIMUM WIRE LENGTH

R. A. Rutman
AC Spark Plug Division
General Motors Corporation
El Segundo, California

INTRODUCTION

The efficient placement of components has taken on increased importance because of microminiaturization techniques. Packaging techniques are permitting higher density of components. As the components are mounted closer and closer, less and less room becomes available between components for interconnections. Under these circumstances, the layout of circuitry becomes a difficult and time consuming task for the designer. One approach often used by designers is to group together elements which are functionally related so as to reduce the length of interconnections. If the designer could place the elements in such a way that every element connects only to its nearest neighbors, the problem of routing the interconnections would be somewhat simplified.

In general, the shorter the wires the easier they are to lay out. Furthermore, in the special case of a printed circuit board, there is a limit to the total length of etched wires that can be printed on one layer.

Those considerations led to the concept that a placement of components which will minimize the total length of the wires would simplify the layout. With that goal in mind, this paper describes a technique for the automatic placement of components.

1. *Description of Problem*

The motivation for attacking the problem of placement arose from the author's association with a computer development project at AC Spark Plug. In that project, the logic mechanization of a computer used micrologic elements mounted on multi-layered boards. Several layers of etched circuitry provided the necessary interconnections.

Although the work done on this problem was for a specific application, it is felt that some of its features are usable in broader applications. Consequently, definition of some fundamental terminology is hereby given and generalized as much as possible.

A board is defined as a finite array of positions. A position is defined as a point in an x-y coordinate system. Elements are movable entities which may be placed in the given positions.

In these terms the problem is to place a given set of elements into positions on a board so as to minimize the total length of the interconnecting wires.

A major portion of this paper deals with the application where only one element per position is allowed. The technique for allowing more than one element per position is given in Section 7.

2. *The Technique of Solution*

a. Steinberg's Algorithm[1]

A set of elements which have interconnec-

tions determined by the electrical design is given. Assume these elements are placed on the board in some initial positions with an initial wire length. Furthermore, assume that some of these elements are not directly connected together. In fact, unconnected sets of elements form an important part of the solution. The technique of solution consists of selecting unconnected sets and moving the elements within each set to reduce the wire length.

An unconnected set can be formed in the following manner. First select any element on the board. Then choose any element which is not directly connected to this first element and add it to the set. Then choose a third element which is not connected to the first two elements, etc. The set is called maximal when the selection is carried to the point where no more elements can be added. Clearly there are many ways in which maximal unconnected sets can be formed. In particular, a family of unconnected sets can be formed such that every element is in at least one set. Section 5 contains more details on the formation of unconnected sets.

Starting with an initial placement of elements in board positions, take some unconnected set of the elements (possibly a maximal unconnected set) and remove it from the board. A set of positions on the board are now vacant. If there were V vacant positions before removal of the unconnected set, now there would be V + N vacant positions, where N is the number of elements in the unconnected set.

Next, take one element from the unconnected set and place it in one of the vacant positions. The total length of the wires which are attached to the element in this position can be determined. This length is independent of the placement of any of the other elements in the unconnected set, since the given element is not connected to any of them. Now place this element in each of the vacant positions and determine the wire length for each position. Then do the same for each element in the unconnected set. This gives an array of N (N+V) values.

The next step is to determine a placement of the unconnected set in the vacant positions so that the total wire length for all the elements in the unconnected set is as small as possible. To do that there exists an algorithm which guarantees a minimum placement of the unconnected set. See Section 6 for details.

Notice that the total wire length *between* elements which are *not* in the unconnected set remains constant during this manipulation since they remain fixed on the board.

After manipulation of this unconnected set, the worst that can happen is that there is no change in the total wire length (perhaps, all elements in the unconnected set end up where they were originally). However, if there exists a better placement of the unconnected set, the algorithm will find it, and the total wire length will decrease.

The process which has just been described will be referred to as an iteration. The next step in the algorithm is to choose another unconnected set and repeat the process. The algorithm continues through many such iterations until the total wire length no longer decreases or decreases an insignificant amount.

b. Improvements on Steinberg's Algorithm

(1) Improvement in the method of measuring wire length. The original Steinberg algorithm uses a connection matrix (C_{ij}). For example, if element e_i is connected to element e_j by three wires, term C_{ij} of the connection matrix is equal to three. This method of expressing connection with integers is perfectly adequate in some cases. But in other cases, a fractional description of the connection is desirable. The author's interpretation of Steinberg's algorithm based upon examples[1] is that C_{ij} assumes only integral values. Consider the example in Figure 1. If only integers are permitted, these two different connections would be represented by the same values in the connection matrix and hence would be treated identically in the placement process.

For the measurement of wire length, the placement program described in this paper

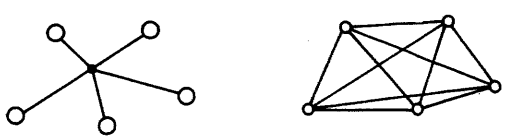


Figure 1.

makes a distinction between these two cases and is therefore able to converge on a more satisfactory solution for certain types of connections. The measurement of wire length is treated in more detail in Section 4.

(2) Interchange. The original algorithm provides a powerful method for manipulating elements which are not connected together. Sometimes elements in connected sets can be manipulated to advantage. Consider the example in Figure 2.

If only the transposition A⟷B could occur, a more desirable placement would be obtained. The basic algorithm allows interchange only of elements which are in the same unconnected set. Because A and B may never be in the same unconnected set (since they are connected), the basic algorithm would not resolve this difficulty. In order to improve the basic algorithm, the placement program also evaluates the interchange of all elements which are directly connected together.

(3) Reducing the length of the longest wires. In the basic algorithm, groups of elements which are tightly connected to one another are reluctant to move.

This is best seen by example. Suppose the placement problem consists of three elements, A and B movable, and X fixed (X might be a connector pin, for example), connected and placed as shown in Figure 3(a). Assume lots of vacant positions. Suppose B is in the first unconnected set. Then, at the end of the first iteration, the result would be as shown in Figure 3(b).

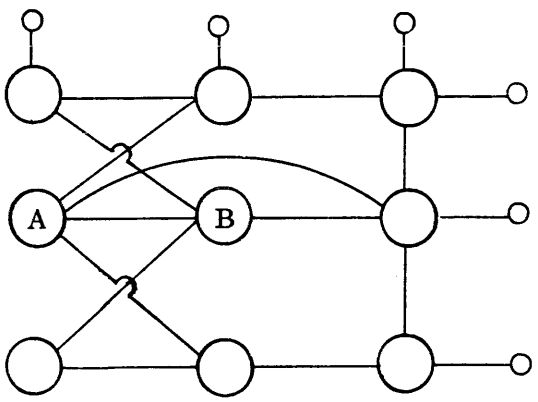


Figure 2.

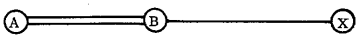


Figure 3(a).

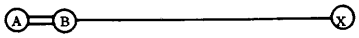


Figure 3(b).

Figure 4.

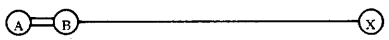


Figure 5(a). Initial State.

If A is in the second unconnected set, A can wander into any one of four positions about B where the total wire length is equivalent. The difficulty is that the group (A, B) will not tend to migrate toward the fixed element X. So minimum total wire length is not reached! The element X need not be fixed. For example, the group A, B, C, and D shown in Figure 4 are all movable, yet will not come together unless by chance.

A partial solution to this problem is obtained in the following way. The algorithm is first applied to the task of minimizing the lengths of the longest wires, and later it is applied to minimizing total wire length. The former task is referred to as Phase I of the placement program, and the latter is referred to as Phase II. The diagrams in Figure 5 illustrate what happens to the example given in Figure 3 as attempts to minimize the lengths of the longest wires are made in a Phase I placement.

Note that in this process the total wire length sometimes increases. In the example, Phase I alone converged to the minimum wire length and Phase II would produce no further improvement. In general, however, Phase II is needed

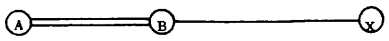


Figure 5(b). B belongs to first unconnected set and moves.

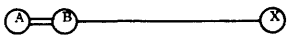


Figure 5(c). A belongs to second unconnected set and moves.

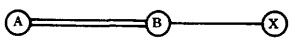


Figure 5(d). B moves.

Figure 5(e). A moves. The result is one of the four possible permutations of A.

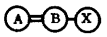


Figure 5(f). B moves.

to make the fine adjustments; Phase I makes the coarse adjustments. The exact function used to measure the length of the longest wire in Phase I is described in detail in Section 4.

3. *Computer Results and Running Times*

To test these techniques the placement program was run using two computer boards. The first, referred to as the small board, contained 70 elements and 77 positions. The second, referred to as the large board, contained 516 elements and 550 positions. To evaluate the results of each placement, the minimum point to point wire length was computed using the algorithm of Loberman and Weinberger.[4]

In the following discussion, an initial placement is described as either a *random* initial placement or a *manual* initial placement. The former was determined by use of a random number table. The latter was prepared by a skilled designer.

Because of the larger amount of computer time required to place the large board, the small board was used for most of the comparisons.

a. The Results of the Original Steinberg Algorithm

The original Steinberg algorithm[1] when applied to the small board with a manual initial placement resulted in a 4.7% reduction in total wire length.

b. The Effect of Different Wire Length Measurement

If every pin were connected to only one other pin, the method of measuring wire length described in this paper would be identical to that of Steinberg's. If a large percentage of the pins are connected to more than one other pin, the method in this paper would have a pronounced effect. In the particular case of the small board, a majority of the pins are connected to only one other pin. Thus the percentage of improvement over the manual initial placement is only 5.3%. That is, if the original Steinberg algorithm is used with no change other than in the measurement of wire length, the improvement is 5.3%.

c. The Effect of Interchange

Investigation is still underway to determine how often an interchange should be performed. Possibly an interchange should be done every iteration.

As mentioned above, there is a 5.3% improvement over the manual placement with no interchange. With an interchange once every 20 iterations, there is a 6.7% improvement. With an interchange every 10 iterations, there is an 8.2% improvement. The improvements were measured using the Loberman and Weinberger technique.[4] Figure 6 illustrates the effect of interchange upon reducing pseudo wire length. Two runs were made on the small board. One run was without interchange and one was with interchange every 20 iterations.

d. The Effect of First Reducing Longest Wire Length

Starting with a manual placement of the small board and going through Phase I and Phase II with an interchange every 10 iterations, the result is a 19.4% improvement over the manual placement.

Starting with a random initial placement and going through Phase I and Phase II with an interchange every 20 iterations, the result is a 17.8% improvement over the manual placement.

e. Improvement on the Large Board

Starting with a manual initial placement of the large board and going through Phase I and

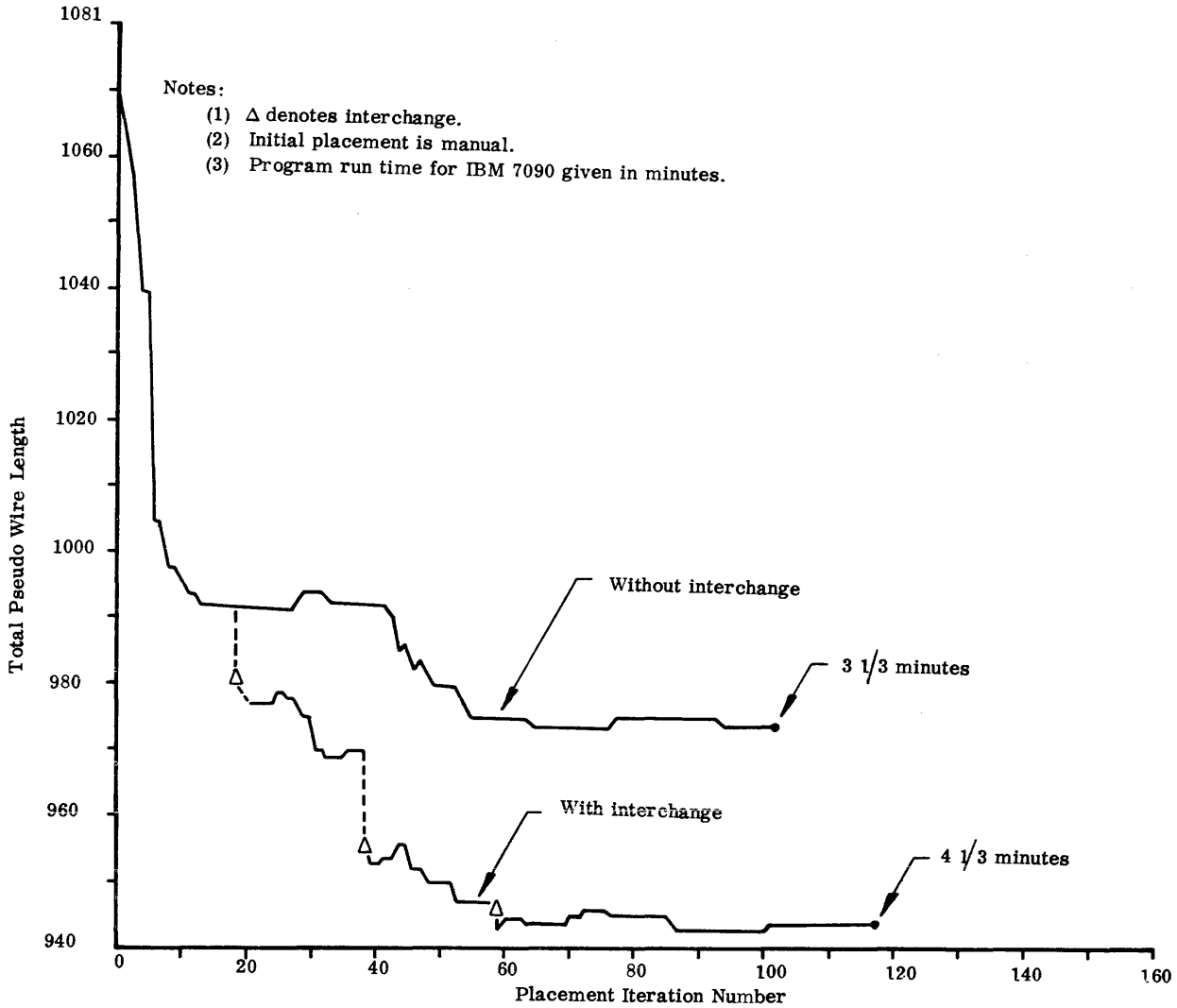


Figure 6. Small Board (Phase II).

Phase II with interchange approximately every 10 iterations, there was a 39% improvement over the manual placement.

f. Computer Running Times

For the placement of the small board, the computer was permitted to run to completion and required a total of approximately 4 minutes of IBM 7090 computer time. Notice that most of the improvement occurs in the first few iterations. In the placement of the large board, there was some question as of when to cut off execution of the program. The program was run for 2 hours 10 minutes in Phase I and for 2 hours in Phase II. However, both phases could have been stopped sooner with little loss in improvement. This is clearly illustrated in Figures 7 and 8.

4. *The Technique for Measuring Wire Length*

The length of wire needed to interconnect a set of points depends upon the manner in which the routing is done. For a simple connection, say between point A and point B, it is assumed that the length of a connection is the dog leg distance between A and B. This is adequate for simple connections between two points, but as the number of points gets greater the determination of wire length becomes more and more complex and depends largely on the specific point to point wiring scheme. For example, consider the case of five points A, B, C, D, and E

connected by one conductor as shown in Figure 9.

Each of the routings in Figure 9 has a different wire length. Steinberg,[1] in placing element E, would treat the wire length as in routing 2. In treating element A, he would treat the wire length as in routing 4.

The problem is still more complicated for printed circuits since connections can be made from point to wire in addition to point to point as shown in Figure 10.

There does not seem to be a simple technique for computing the minimum wire length for all connections of this type.

Loberman and Weinberger[4] have an algorithm for computing the minimum point to point wire length, but it is too slow to use in the placement program.

The placement program for the most part does not have to evaluate the actual wire length. But it must provide the answer to such questions as the following.

If elements A, B, C, D, and E are connected together by one conductor and if A, B, C, D are fixed, with E able to occupy either position x or position y, what is the difference in the total wire length of the conductor if E is in position x compared to that of position y? Hence, of principal concern is the change in wire length from

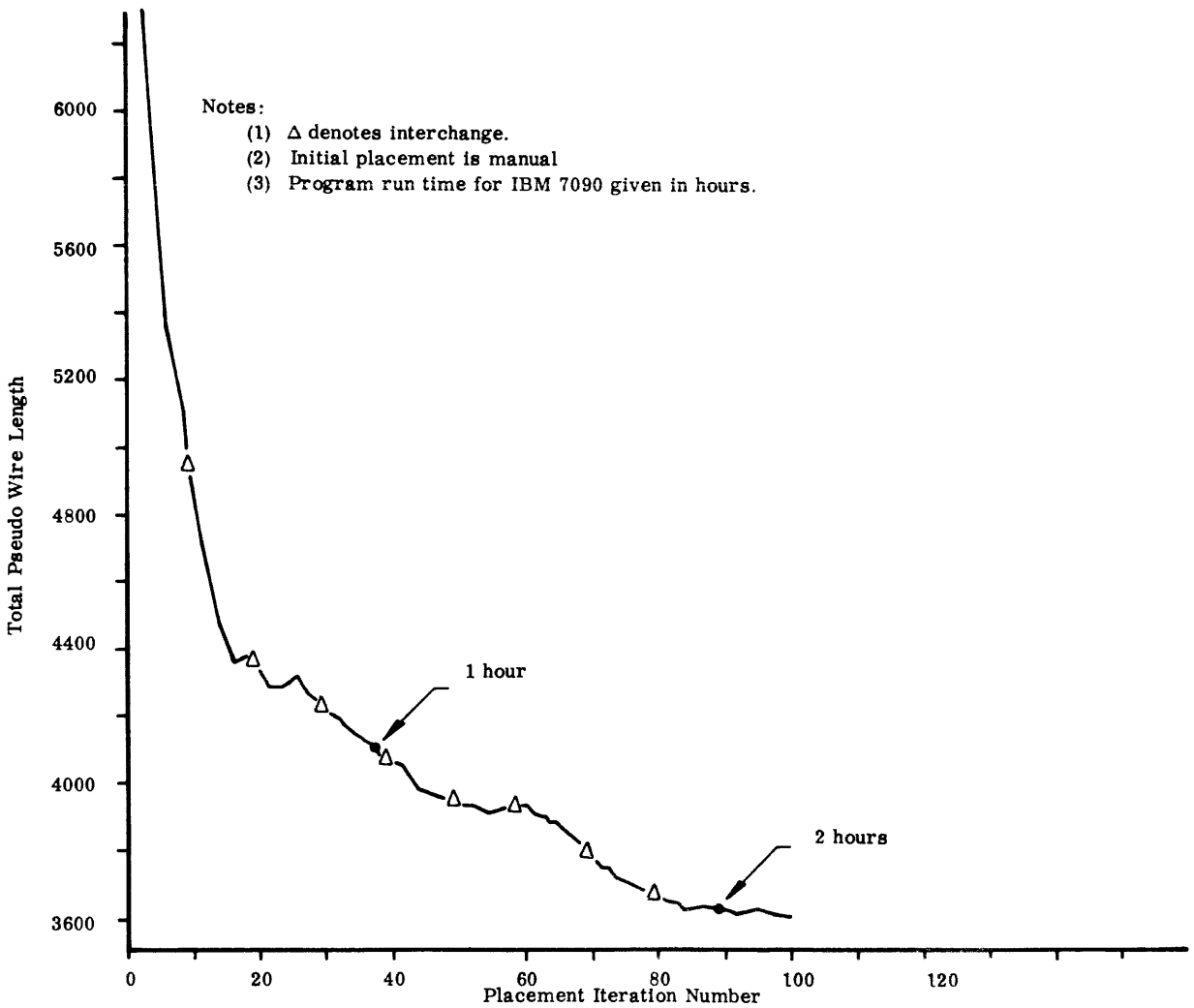


Figure 7. Large Board (Phase I).

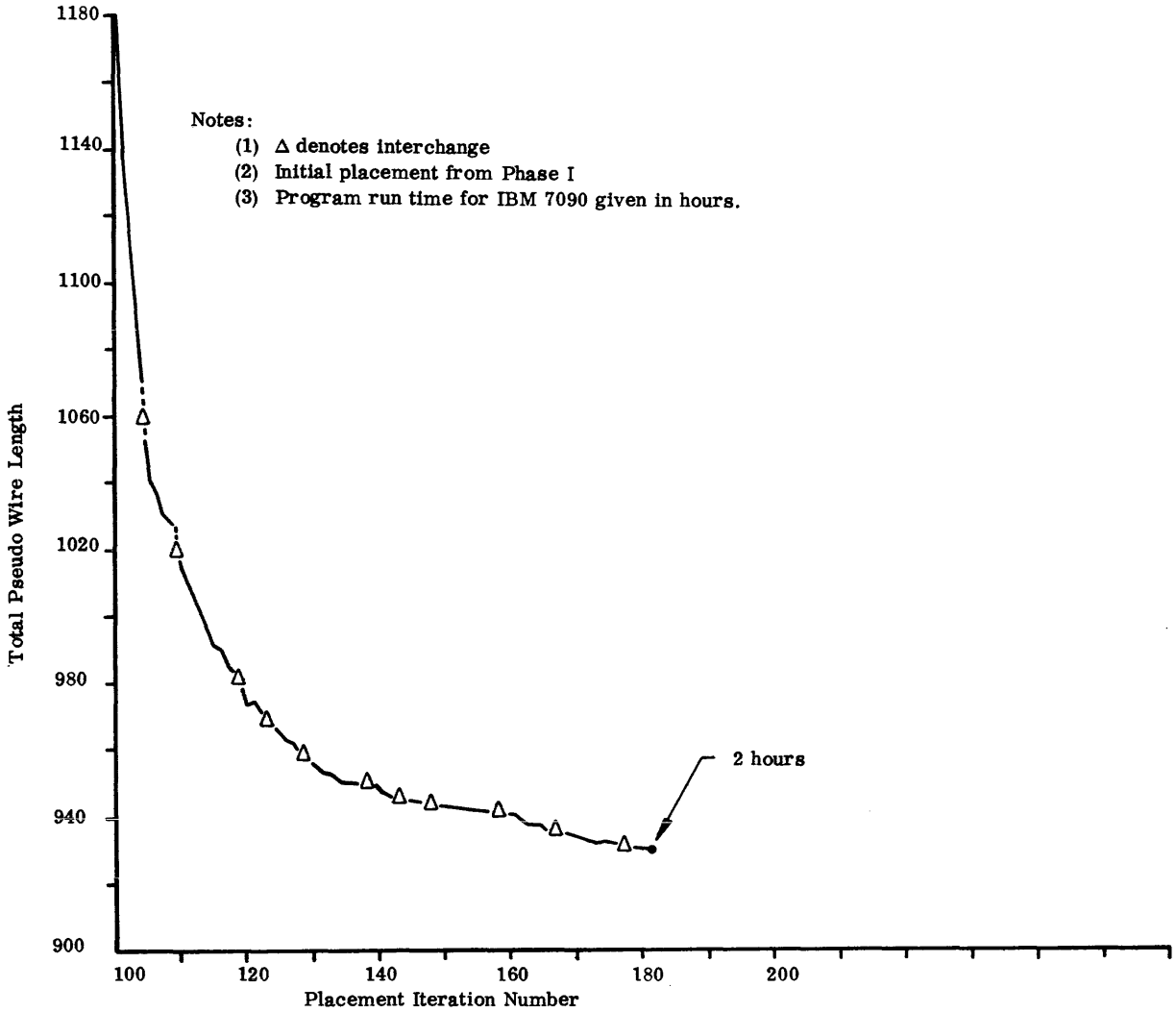


Figure 8. Large Board (Phase II).

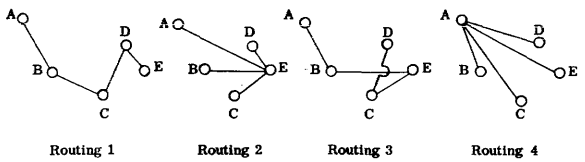


Figure 9.

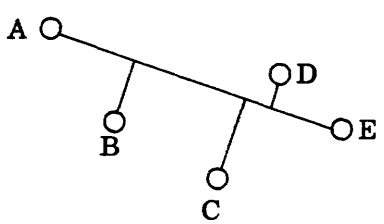


Figure 10.

one position to another rather than in its absolute value.

The remainder of this section deals with the exact function used to evaluate wire length.

Two pins are defined to be connected if they are electrically connected. The fact that pin A is connected to pin B forms a relation between pin A and pin B. It is easy to verify that this is an equivalence relation. Elements e_i and e_j are said to be connected if there exists a pin ϵ e_i and a pin $q \epsilon e_j$ such that pin p is connected to pin q. Consider the set of all pins in a given problem. Let connection be the equivalence relation on this set. This equivalence relation will divide the set of all pins into disjoint equivalence

classes. Each of the equivalence classes so formed is called a conductor.

Let $S = \{ e_1, e_2, . , e_i, . e_f\}$ be the given set of f elements.

Let $L = \{ \ell_1, \ell_2, \ell_3, . , \ell_n, . , \ell_g\}$ be the given set of g positions (locations) where $g \geq f$.

Let $P = \{ p_1, p_2, \ldots , p_m, p_s\}$ be the set of all pins on all elements.

Let $C = \{ c_1, c_2, . , c_j, . , c_h\}$ be the set of conductors.

Each conductor is a subset of the set P. Also, each element e_i is a subset of the set P. Pin $p_m \epsilon e_i$ if pin p_m is a pin on element e_i.

Now define the notion of a transductor T_j which is a set of elements connected together by the conductor c_j.

$$T_j = \{ e_i \mid p_m \epsilon c_j \text{ and } p_m \epsilon e_i\}.$$

Notice that there are exactly as many transductors as conductors and that there is a natural mapping from conductor to transductor. Given a conductor c_j, merely consider each $p_m \epsilon c_j$ in order. Given a p_m, there exists an e_i such that $p_m \epsilon e_i$. Therefore, the corresponding e_i belongs in the set T_j. Let $\alpha(T_j)$ be defined as a function which maps T_j into its cardinal number, the number of elements in the set.

Also define a one to one function γ whose domain is the set S of elements and whose range is the set L of positions. $\gamma(e_i)$ is the present location of element e_i. Each position in the set L has two components: the x coordinate and the y coordinate. Thus, $\gamma_x(e_i)$ is the x coordinate of element e_i, and $\gamma_y(e_i)$ is the y coordinate of element e_i.

The first step in the placement process is to assign all elements to random initial positions. After removing an unconnected set from the board there is a set M of available positions. M is a subset of L. Suppose e_i is in the unconnected set. At this stage of the placement process, e_i may be placed in any position of the set M. The placement process uses a number called the pseudo wire length $F_{i,n}$ for each position $\ell_n \epsilon M$ which indicates how well element e_i likes to be in position ℓ_n (assuming every element that e_i is connected to is not movable). A comparatively low number means that the wire length is low when e_i is in position ℓ_n. A comparatively high number means that the wire length is high when e_i is in position ℓ_n. Phase I and Phase II (as defined in Section 2) use different functions for the pseudo wire length. The functions are referred to as $F^{I}_{i,n}$ and $F^{II}_{i,n}$, respectively.

At each step in the placement process the total pseudo wire length can be evaluated as follows. Suppose the current placement puts element e_i in position $\ell_{p(i)}$. Then the total pseudo wire length is defined as $\sum_i F_{i,p(i)}$. The object of the program is to find a placement which will minimize $\sum_i F^{II}_{i,p(i)}$.

Abstractly, $F_{i,n}$ is a function of $\gamma(e_k)$ and $\gamma(e_i)$ for all k such that $e_k \epsilon T_j$ and $k \neq i$ and all j such that $e_i \epsilon T_j$ and $\alpha(T_j) > 1$. Element e_i is assumed to be in position ℓ_n, having x, y coordinates ℓ_{nx} and ℓ_{ny}.

$F^{II}_{i,n}$ the Phase II pseudo wire length number for element e_i when in position ℓ_n, is given by:

$$F^{II}_{i,n} \equiv \sum_{\text{all } j} \frac{1}{\alpha(T_j) - 1} \sum_{\text{all } k} (|\gamma_x(e_k) - \ell_{nc}| + |\gamma_y(e_k) - \ell_{ny}|)$$

where $e_i \epsilon T_j$, $e_k \epsilon T_j$ $k \neq i$, and $\alpha(T_j) > 1$.

To get an expression for $F^{I}_{i,n}$ the phase I pseudo wire length number, it is necessary first to define an auxiliary function $F_{i,n}^{*j}$ for three separate cases.

Case 1: $\alpha(T_j) = 2$

$$F_{i,n}^{*j} \equiv |\gamma_x(e_k) - \ell_{nx}| + |\gamma_y(e_k) - \ell_{ny}|$$

where $T_j = \{ e_k, e_i\}$.

Case 2: $\alpha(T_j) = 3$

$$F_{i,n}^{*j} \equiv \max \{ (|\gamma_x(e_k) - \ell_{nx}| + |\gamma_y(e_k) - \ell_{ny}|), (|\gamma_x(e_m) - \ell_{nx}| + |\gamma_y(e_m) - \ell_{ny}|) \}$$

where $T_j = \{ e_m, e_k, e_i\}$.

Case 3: $\alpha(T_j) > 3$

For this case a pseudo location ℓ_n^{*j} is first defined, its x and y components given by:

$$\ell_{ix}^{*j} \equiv \frac{1}{\alpha(T_j)} \sum_k \gamma_x(e_k) \text{ where } e_k \epsilon T_j \text{ and } k \neq i.$$

$$\ell_{ix}^{*j} \equiv \frac{1}{\alpha(T_j)} \sum_k \gamma_y(e_k) \text{ where } e_k \epsilon T_j \text{ and } k \neq i.$$

Furthermore, let a function G_i^{*j} be given by:

$$G_i^{*j} \equiv \frac{1}{\alpha(T_j) - 1} \sum_{\text{all } k} (|\gamma_x(e_k) - \ell_{ix}^{*j}| + |\gamma_y(e_k) - \ell_{iy}^{*j}|)$$

where $e_k \epsilon T_j$ and $k \neq i$.

Then $F_{i,n}^{*j}$ can be expressed as:

$$F_{i,n}^{*j} \equiv \left[\frac{1}{\alpha(T_j) - 1} \sum_{\text{all } k} \left(|\gamma_x(e_k) - \ell_{nx}| + |\gamma_y(e_k) - \ell_{ny}| \right)\right] - G_i^{*j}$$

where $e_k \epsilon T_j$ and $k \neq i$.

Now the phase I measure $F_{i,n}^{I}$ for wire length is defined as:

$$F_{i,n}^{I} \equiv \max \{F_{i,n}^{*j}\}$$

where $e_i \epsilon T_j$ and $\alpha(T_j) > 1$.

One of the advantages of choosing the functions in this manner is the ease of computation. Another is that they tend to closely approximate the real situation. As an example, consider the element e_i which has only one transductor T_j. Suppose $\alpha(T_j) = 2$. Then $F_{i,n}^{II}$ is merely the dog leg distance between the two elements contained in T_j.

Suppose e_i is contained in two transductors T_1 and T_2 such that $\alpha(T_1) = \alpha(T_2) = 2$, $T_1 = \{ e_k, e_i \}$, and $T_2 = \{ e_m, e_i \}$. That is, e_i is connected to two different elements as shown in Figure 11.

Then $F_{i,n}^{II}$ would be equal to seven which is the length of the longest wire. Suppose we also had $T_3 = \{ e_m, e_i \}$ with $\alpha(T_3) = 2$ as shown in Figure 12.

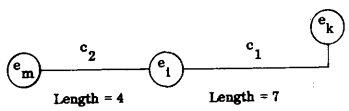


Figure 11.

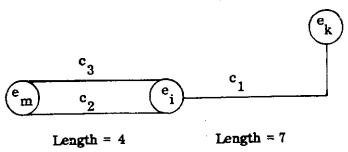


Figure 12.

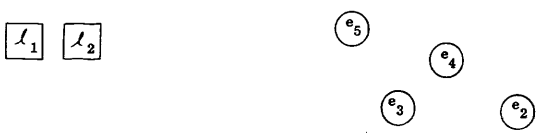


Figure 13.

Here again $F_{i,n}^{I} = 7$ since $F_{i,n}^{*3} = F_{i,n}^{*2} = 4$ and $F_{i,n}^{*1} = 7$. $F_{i,n}^{II}$ for the case in Figure 11 would be equal to 11, the length of both wires.

$F_{i,n}^{II}$ for the case in Figure 12 would be equal to 15.

For another example, suppose e_1 belongs only to transductor T_j where $\alpha(T_j) = 5$ and $T_j = \{ e_1, e_2, e_3, e_4, e_5 \}$. Then $F_{i,n}^{II}$ can be interpreted as the average distance from element e_1 to the other elements in T_j. Suppose $M = \{ \ell_1, \ell_2 \}$ where ℓ_1 and ℓ_2 are one unit apart, and both positions are at some distance from the other elements in T_j as shown in Figure 13.

Then $F_{1,1}^{II} - F_{1,2}^{II} = 1$ which is desirable because the wiring would be done in a manner similar to Figure 14 below, and the difference in actual wire length between e_1 in ℓ_1 and e_1 in ℓ_2 is only 1 unit.

Suppose positions ℓ_1 and ℓ_2 and elements e_2, e_3, e_4, and e_5 are located as shown in Figure 15 below, that is, interior to the set.

Then $F_{1,1}^{II} - F_{1,2}^{II} = 0$. That is, the two positions are equivalent and e_1 may be placed in either position with no difference in wire length. Once again the function approaches the real situation.

For the case shown in Figure 5, $F_{1,1}^{I} = F_{1,2}^{I} = 0$. $F_{1,1}^{I}$ would be greater than zero only if ℓ_1 were a position located outside the rectangle formed by e_2, e_3, e_4, and e_5 as in Figure 11. For Figure 11, $F_{1,1}^{I} - F_{1,2}^{I} = 1$ as would be expected.

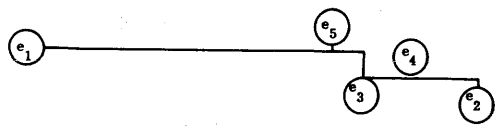


Figure 14.

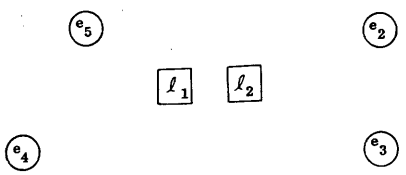


Figure 15.

In many problems, the majority of the transductors are such that $\alpha\ (T_j) = 2$. But the case where $\alpha\ (T_j) > 2$ must be handled consistently. Furthermore, it is observed that given any element e_i there is a large probability that $e_i \epsilon T_j$ such that $\alpha\ (T_j) > 2$. Table 1 shows the result of a survey made on a large computer printed circuit board. The board had 729 transductors with the distribution as shown. N is the number of transductors with the given $\alpha\ (T_j)$. In other words, the sets T_j were sorted into groups according to their cardinal number.

5. *The Procedure for Forming Maximal Unconnected Sets*

Given a set S of elements e_i and a set of transductors (a typical transductor being T_j) an unconnected set U is defined as follows:

$U = \{ e_i, e_k \epsilon S \mid$ there is no T_j such that $e_i \epsilon T_j$ and $e_k \epsilon T_j \}$.

There are many ways in which the set U can be formed. It is desirable to form a family $U' = \{ U_1, U_2, U_3, \ldots, U_r, \ldots U_q \}$. Each member of S is contained in at least one of the sets in U'.

Also, it appears desirable that the unconnected sets should overlap one another.[1] In dealing with a large number of elements it is not feasible to form all unconnected sets. Instead a technique is used which causes some overlap. About ¼ of the elements in each set will be contained in other sets. Figure 16 is a flow chart which shows the technique for obtaining the unconnected set U_j given the sets $U_0, U_1, U_2, \ldots, U_{j-1}$. The following nomenclature is used in Figure 16.

E_0 = The original set of elements which are eligible for inclusion in an unconnected set.

E = A working set containing elements eligible for inclusion in the *current* unconnected set.

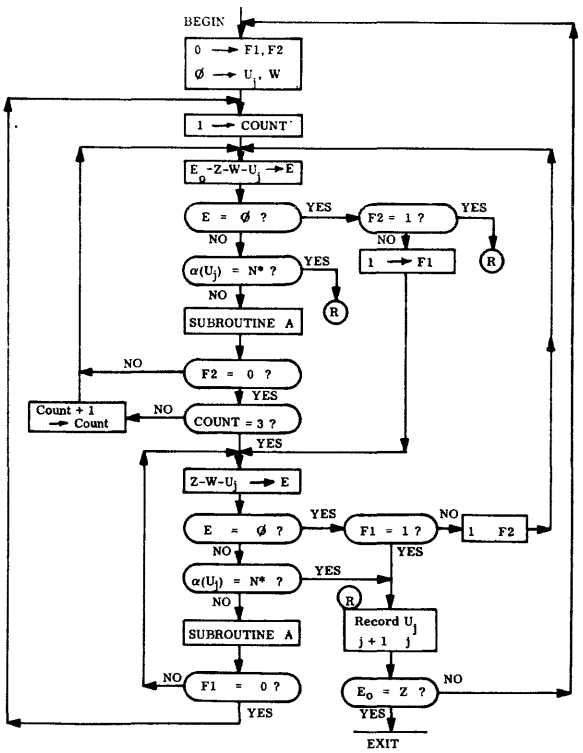
W = A working set containing some of the elements not eligible for inclusion in the current unconnected set. Elements in W are connected to elements which are in the current unconnected set U_j.

U_0 = Φ, the void set.

$\alpha(U_j)$ = Cardinal number of set U_j.

N* = The largest number of elements which may be contained in any unconnected set. This number is a function of the number of empty positions on the board and the amount of avialable space in the computer. In short, a given placement program is restricted as to the size of the solution matrix which it can solve. By restricting the number of elements in an unconnected set, the size of the matrix is restricted. This means that the sets may not be maximal in some cases.

$Z = \bigcup_{m=1}^{j-1} U_m$ (the union of sets $U_1, U_2, \ldots, U_{j-1}$).



NOTES:
(1) Subroutine A consists of Random selection of an e_i from E, placing it in U_j, and placing all e_k connected to e_i in W.
(2) Given sets A and B then A-B is defined as the set of elements in A but not in B.

Figure 16. Flowchart for Formation of Unconnected Sets.

NOTES:

(1) Subroutine A consists of Random selection of an e_i from E, placing it in U_j, and placing all e_k connected to e_i in W.

(2) Given sets A and B then A-B is defined as the set of elements in A but not in B.

6. *The Solution Matrix*

Each iteration of the placement program involves the determination of the optimal placement for an unconnected set of elements. The procedure for determining an optimal placement involves operations on a solution matrix described in this section. The method for manipulating the matrix is an adaptation of an algorithm by Munkres.[2]

The first step in placing an unconnected set is to form a matrix in the following manner. Suppose the unconnected set has N elements and there is a set of N + V positions available for placement. Let the set of available positions be indexed by n, with n running from 1 to N + V. Let the unconnected set of elements be indexed by i with i running from 1 to N. Now form a matrix [A] of N rows and N + V columns. Each row i of the matrix corresponds to the element e_i in the unconnected set and each column n of the matrix corresponds to the position l_n in the set of available positions. The elements $a_{i,n}$ of the matrix are set equal to the functions $F^{II}_{i,n}$ (or $F^{I}_{i,n}$ if in phase I of the placement) defined in Section 4 on measuring wire length.

The Munkres algorithm[2] results in N stars (asterisks) being associated with elements of the matrix [A]. These stars are positioned so that every row has exactly one star and no column has more than one star. The placement of the stars in the matrix describes the placement of the elements e_i. Thus if element $a_{3,2}$ of the matrix has a star upon completion of the algorithm, then element e_3 should be placed in position l_2 in the set of available positions.

The following algorithm is derived from Munkres' algorithm. It can handle rectangular matrices and includes features to speed up the computer solution.

During the course of the algorithm, some zero-elements are distinguished by asterisks and some by primes. These are referred to as starred zeros and primed zeros respectively.

In addition to the matrix [A], the row vector R_s is required. Each element of R_s corresponds to a column of the matrix [A]. If the column contains a zero star, the corresponding element of R_s will contain the row number of the zero star. If the column contains no zero star, the corresponding element of R_s will be zero.

Similarly, there are two column vectors C_s and C_p, whose elements correspond to the rows of the matrix [A]. If the row has a zero star, the corresponding element of C_s contains the column number of the zero star. If the row has a zero prime, the corresponding element of C_p contains the column number of the zero prime.

A "line" is defined as a row or a column of the matrix [A]. In the course of the algorithm, certain lines are distinguished and referred to as "covered" lines. An element of the matrix is said to be uncovered, once-covered, or twice-covered, accordingly as it lies in precisely none, one, or two covered lines.

There is the column vector C_c, each element of which corresponds to a row of the matrix. If the row is covered, the corresponding element of $C_c = 1$, otherwise it equals 0. Each element of the row vector R_c corresponds to a column of the matrix. If the column is covered, the corresponding element of $R_c = 1$, otherwise it equals 0.

The coordinates of uncovered zeros in the matrix are contained in so-called zero lists so that they may be quickly located without testing each element of the matrix. Zero list A contains the coordinates of all uncovered zeros and the coordinates of some zeros which may or may not be covered. Zero list B contains the coordinates of some zeros which are currently covered but which may at some future time become uncovered. Zero list A and zero list B are push down lists[6] with a bottom pointer. That is, there is a cell pointing to the bottom cell of the list. In step 2, zero list B is emptied and added to the bottom of zero list A. This operation is facilitated by the bottom pointer in zero list A.

Preliminaries. In the matrix [A], no lines are covered and no zeros are starred or primed. Consider row 1 of the matrix. Find the smallest element in row 1, and call it h. Subtract h from each element of row 1. In the process of subtracting h, some element (or elements) of row 1 become zero. Whenever an element becomes zero and if there is no starred zero in the row and none in its column, star the zero and cover the column containing the zero. If there is no zero star in its column but there is one in its row, add the matrix coordinates of the zero to

the zero A list (unless the zero A list is already full). Do the same for each row of the matrix.* If N columns are covered, the starred zeros form the desired result, and the problem is finished. If less than N columns are covered, go to step 1.

Step 1

Take the coordinates of a zero from the zero A list, popping up the zero A list. If the zero is covered, push the coordinates of the zero into the zero B list and take another zero from the zero A list. Continue until a non-covered zero is found or until the zero A list is depleted. If the zero A list is depleted and flag F is set, search the matrix for an uncovered zero. If the zero A list is depleted and flag F is reset, go to step 3.

Assume an uncovered zero is found. Prime the zero, then consider the row containing it. If there is no starred zero in this row, go at once to step 2. If there is a starred zero in the row, cover the row and uncover the column of the starred zero. Look for any uncovered zeros in this new uncovered column. If any exist, add their coordinates to the top of the zero A list (if the zero A list overflows, set flag F). Then go to beginning of step 1.

Step 2

There is a sequence of alternating starred and primed zeros, constructed as follows: Let Z_0 denote the uncovered 0′ (there is only one). Let Z_1 denote the 0* in Z_0's column (if any). Let Z_2 denote the 0′ in Z_1's row (if Z_1 exists, its column is not covered since Z_0 is in its column and Z_0 is not covered, so its row must be covered. Hence there is a 0′ in this row (see step 1). Let Z_3 denote the 0* in Z_2's column (if any). Similarly, continue until the sequence stops at a 0′ which has no 0* in its column (Munkres proves such a 0′ exists and the sequence is unique)).

Now unstar each starred zero of the sequence, and star each primed zero of the sequence. Erase all primes (that is, remove the zero prime coordinates from vector C_1). Uncover every row and cover every column containing a 0*. If N columns are covered, the starred zeros form the desired result and the algorithm is finished. Otherwise, empty the zero B list into the bottom of the zero A list and go to step 1.

Step 3

Let h denote the smallest uncovered element of the matrix; it will be positive. Add h to each twice-covered element. Subtract h from each uncovered element. In the process of subtracting h, add the coordinates of any new zeros to the zero A list. Return to step 1 without altering any stars, primes, or covered lines.

For an example consider the board indicated in Figure 17.

Also assume a set of six elements is given and connected as shown in Figure 18.

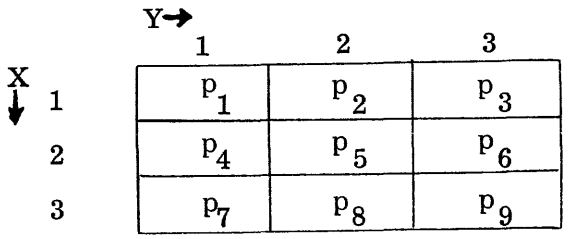


Figure 17.

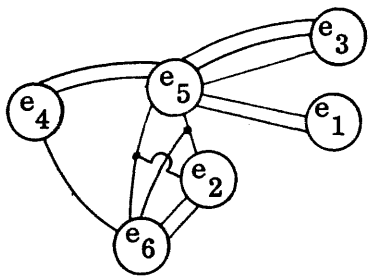


Figure 18.

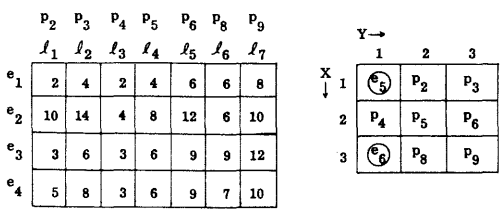


Figure 19.

* Note that at this time the zero A list (if sufficiently large) will contain the coordinates of every uncovered zero if any exist. If there are too many zeros for the zero list to hold, flag F will be set. Otherwise, flag F is reset.

Suppose element e_5 is fixed in position p_1 and element e_6 is fixed in position p_7. Then the set of available positions is { p_2, p_3, p_4, p_5, p_6, p_8, p_9}. Choose the unconnected set consisting of elements e_1, e_2, e_3, and e_4. For this choice form the matrix A whose elements are $F^{II}_{i,n}$. The results are given in Figure 19.

For example, the computation of $F^{II}_{2,5}$ involves element e_2 and position p_6 ($x = 2, y = 3$). Element e_2 belongs to four transductors and is connected to element e_5 (at $x = 1, y = 1$) and element e_6 (at $x = 3, y = 1$). Hence $F^{II}_{2,5}$ is computed as follows:

$$F^{II}_{2,5} = \frac{1}{3-1}[(|1-2|+|1-3|)+(|3-2|+|1-3|)]$$
$$+\frac{1}{3-1}[(|1-2|+|1-3|)+(|3-2|+|1-3|)]$$
$$+\frac{1}{2-1}[|3-2|+|1-3|]$$
$$+\frac{1}{2-1}[|3-2|+|1-3|]$$
$$=\frac{6}{2}+\frac{6}{2}+\frac{3}{1}+\frac{3}{1}=12$$

To solve the matrix, first do the preliminaries. The result is shown in Figure 20.
Since N = 4 but only two columns are covered, proceed to Step 1.

Step 1 examines zero 1, 3 in the zero A list, but finds that it is covered. Step 1 finding no uncovered zeros transfers to step 3. Step 3 finds that 2 is the smallest uncovered element and proceeds to subtract 2 from each uncovered element and add 2 to each twice-covered element (there are none). The result is shown in Figure 21.

Step 3 transfers back to step 1. Step 1 finds an uncovered zero in 2, 6 and primes it. The row contains a 0* in column 3. Therefore, the row is covered, and column 3 is uncovered. The newly uncovered zeros in column 3 are added to zero list A. The result is given in Figure 22.

The next uncovered zero that step 1 finds is 4, 3. The zero is primed, and since the row contains no 0*, step 1 goes to step 2.

Step 2 constructs the following sequence:

$$0' = 4, 3; 0^* = 2, 3; 0' = 2, 6.$$

Primes of the sequence are changed to stars and stars of the sequence are removed. At the end of step 2, the results are as shown in Figure 23.

Since N = 4 and only three columns are now covered, step 2 returns to step 1. The next uncovered zero that step 1 finds is 1, 4. The zero is primed. Its row has a 0* in column 1. Therefore, the row is covered and column 1 is uncov-

	p_2 (ℓ_1)	p_3 (ℓ_2)	p_4 (ℓ_3)	p_5 (ℓ_4)	p_6 (ℓ_5)	p_8 (ℓ_6)	p_9 (ℓ_7)
e_1	0*	2	0	2	4	4	6
e_2	6	10	0*	4	8	2	6
e_3	0	3	0	3	6	6	9
e_4	2	5	0	3	6	4	7

	C_c	C_s	C_p
	0	1	0
	0	3	0
	0	0	0
	0	0	0

R_c	1	0	1	0	0	0	0
R_s	1	0	2	0	0	0	0

Zero Tables

A		B	
i	n	i	n
1	3		

Figure 20.

	p_2 (ℓ_1)	p_3 (ℓ_2)	p_4 (ℓ_3)	p_5 (ℓ_4)	p_6 (ℓ_5)	p_8 (ℓ_6)	p_9 (ℓ_7)
e_1	0*	0	0	0	2	2	4
e_2	6	8	0*	2	6	0	4
e_3	0	1	0	1	4	4	7
e_4	2	3	0	1	4	2	5

	C_c	C_s	C_p
	0	1	0
	0	3	0
	0	0	0
	0	0	0

R_c	1	0	1	0	0	0	0
R_s	1	0	2	0	0	0	0

Zero Tables

A		B	
i	n	i	n
2	6	1	3
1	4		
1	2		

Figure 21.

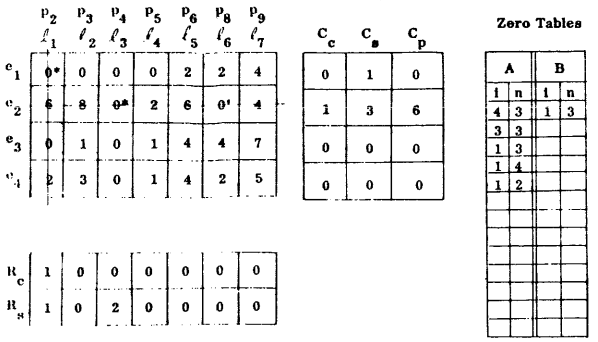


Figure 22.

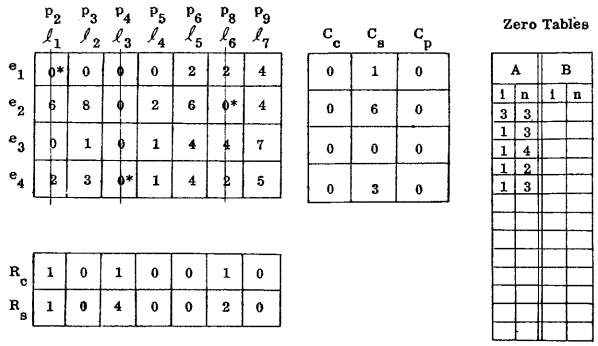


Figure 23.

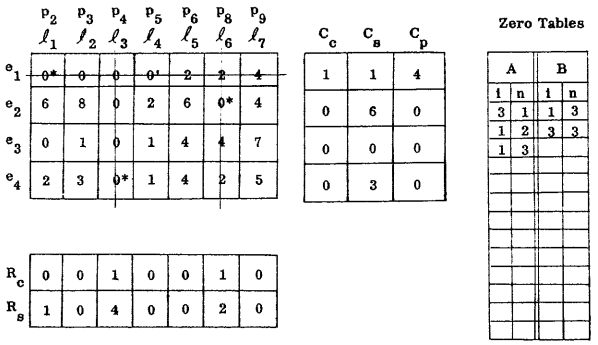


Figure 24.

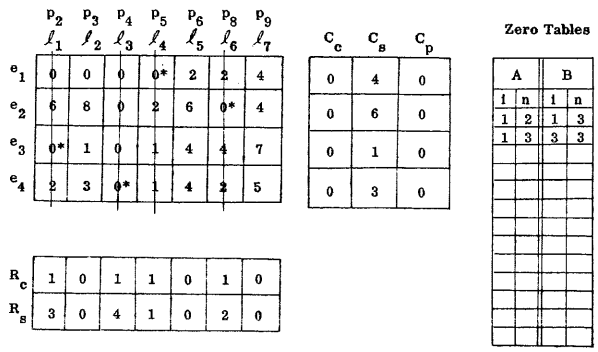


Figure 25.

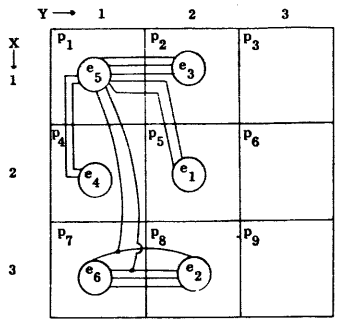


Figure 26.

ered. Then the zero in column 1 is added to the zero A list. The result is shown in Figure 24.

Step 1 then finds the uncovered zero 3, 1 and primes it. Since its row has no 0*, step 1 goes to step 2. Step 2 constructs the following sequence:

$$0' = 3, 1; 0^* = 1, 1; 0' = 1, 4.$$

At the end of step 2, the results are as shown in Figure 25.

Now four rows are covered and the algorithm is done. The placement can be read directly from the column vector C_s or the asterisks in the matrix. Element e_1 is placed into position p_5, e_2 in p_8, e_3 in p_2, and e_4 in p_4. Graphically, the result is as shown in Figure 26.

The result was obtained through the application of only one unconnected set. In practice, several unconnected sets would be used until the placement finally converged.

In summary, the purpose of the various tables and vectors is to increase the speed of computation. The purpose of the zero tables is to enable the computer to find quickly an uncovered zero, without searching the matrix for one. Once the coordinates of a zero (i, n) is obtained from the zero A table, it can quickly be tested to see if it is covered (i.e., check element i of vector C_c and element n of vector R_c). Primes and stars are recorded only in the vectors mentioned, not in the matrix. They are shown in the matrix in the example only to make the example easier to follow. Thus, to erase all primes involves merely setting vector C_p to zero. To uncover a column and to cover a row involves the change of only 1 cell in vector R_c

and 1 cell in vector C_{ij}. Similarly, recording and changing of 0*'s is also a simple operation. Thus, the set up and solution of a matrix containing roughly 140 rows and 170 columns takes an average of less than 1 minute of time on the IBM 7090 computer.

7. *More Than One Element in the Same Position*

Suppose there is a unit called a can which contains more than one logic element. For example, a can might hold 4 flip-flops or 6 gates, etc. The technique for handling a placement for this siutation is considered below.

First, place the elements randomly in the cans according to the rules (e.g., 6 gates to a can). Some cans may be only partly full. Then place the cans on the board in some random initial position. Then make a family of unconnected sets for each type of element when there is more than one element of that type in a can. For example, there may be a family of unconnected sets consisting of only gates of a certain type.

Then, manipulate the families of unconnected sets, not using any available positions other than the positions originally occupied by those elements. The result will be that elements are assigned to the cans more optimally than they were in their original assignment. Now make unconnected sets out of the cans and move the cans (using the empty board positions). Thus there are two types of movement. One is movement of the elements within the cans (keeping the cans fixed). The second is movement of the cans. The placement program would alternate between these two types of movement.

BIBLIOGRAPHY

1. STEINBERG, LEON, "The Backboard Wiring Problem: A Placement Algorithm," *SIAM Review*, Vol. 3, No. 1 (January 1961), pp. 37-50.
2. MUNKRES, JAMES, "Algorithms for the Assignment and Transportation Problems," *Journal for the Society for Industrial and Applied Mathematics*, Vol. V (March 1957), pp. 32-38.
3. KUHN, H. W., "The Hungarian Method for the Assignment Problem," *Naval Research Logistics Quarterly*, Vol. II (March-June 1955), pp. 83-97.
4. LOBERMAN, H., and WEINBERGER, A. "Formal Procedures for Connecting Terminals with a Minimum Total Wire Length," *Journal of the Association for Computing Machinery*, Vol. 4, No. 4 (1957), pp. 428-433.
5. JACOBSON, NATHAN, "Lectures in Abstract Algebra," Vol. 1—*Basic Concepts*, D. Van Nostrand Co., Inc., 1962, pp. 4-5.
6. NEWELL, ALLEN, ed., "Information Processing Language—V Manual" Prentice-Hall, Inc., 1961.
7. GILMORE, P. C., "Optimal and Suboptimal Algorithms for the Quadratic Assignment Problem," *Journal of the Society for Industrial and Applied Mathematics*, Vol. 10, No. 2 (June 1962), pp. 305-313.
8. GLASER, ROBERT H., "A Quasi-Simplex Method for Designing Suboptimum Packages of Electronic Building Blocks," *Proceedings of the 1959 Computer Applications Symposium at the Armour Research Foundation at Illinois Institute of Technology*, pp. 100-111.
9. ALTMAN, G. W., DECAMPO, L. A., and WARBURTON, C. R., "Automation of Computer Panel Wiring," *Transactions of the AIEE*, Vol. 79, Pt. 1 (May 1960), pp. 118-125.

APPLICATION OF AN ASSOCIATIVELY ADDRESSED, DISTRIBUTED MEMORY

G. J. Simmons
Sandia Corporation
Albuquerque, New Mexico

INTRODUCTION

Memory techniques which have so far been developed with computer technology all share limitations imposed by the requirement that a memory address be specified uniquely for storage or retrieval of data. In many problems these addresses are merely "dummy variables" with the only meaningful order being one derived from the content of the data points themselves; examples of such operations are sorting, interpolating and catalog look-up. Largely because of the astronomical proportions of the computing task assumed by some real (and reasonable) problems in these and other closely allied areas, there has been an increasing interest in the type of memory in which information is stored (and retrieved) on the basis of content. Storage systems with this modus operandi are generically classified as "associative memories."[1-6]

There are two fundamentally different conceptions of an associative memory. The first can be regarded as exact (in the same manner that a conventional computer memory is exact) since a normal storage address is generated from an examination of the data to be stored.[7,8,9] In this version, an item for entry is characterized by as many descriptors as possible (or as may be limited by the dimension of the memory) and the entry is stored "at the intersection" of these descriptors. The list of descriptors, if regarded as separate from the data word itself, has been given the name *associative criterion* by Kiseda et al.[10] Obviously, a single storage word may in general have a multiplicity of associative criteria, or alternatively, several words may have a common one. This interpretation of an associative memory is concerned with exact data storage and retrieval (albeit the retrieval request may be inexact and incomplete) and not with "interpolation" between stored data points.

The second conception of an associative memory is concerned almost exclusively with an ability to interpolate between stored data points. This does not imply that exact recall of stored data points is no longer desired, although this capability is generally compromised, but rather that the memory should have the ability to interpolate (or extrapolate) in a field of statistically related memory entries. Biological systems evince both types of memory processes (hence their common classification under the heading of associative memories). Apparently the most important design feature of the biological system is what is termed "distributed memory." Unlike a computer memory where a particular item is stored in a sharply defined location the entries into a biological system are distributed through relatively extended volumes of the neural material, so that even partial extirpation cannot erase a strongly recorded memory trace.[11] The concept of distributed memory has received a great deal of attention in the past few years, primarily because of the pioneer work of F. Rosenblatt on percep-

trons,[12,13,14] and the more recent researches of B. Widrow and J. Angell on Madeline computers.[15,16,17]

ASSOCIATIVE ADDRESSING

If all of the data words which are to be stored in an associative memory could be made available in parallel, then by an examination of the individual associative criteria (descriptor lists or independent variables) a decision could be reached as to which words were to be stored near each other. This decision procedure, based on some measure of nearness between the various associative criteria, is precisely the operation which is carried out in computer sorting, for an example. The reason that such programs are so costly in time and machine operations is that the device must, by a scanning, or moving shuffle, effectively simulate the parallel occurrence of the data words. If an associative addressing scheme is to be effective, it must be able to assign each individual memory entry, as it appears serially, to precisely the memory location to which it would have been assigned had the entire memory fill been available in parallel. It is obvious that if the associative criteria are inadequate for the parallel classification, no serial procedure can do better.

The associative addressing scheme which forms the basis of this paper is extremely simple. In lieu of having all of the data words available in parallel so that their associative criteria can be compared simultaneously, the associative criterion of each new word is compared, as it appears for entry into the memory, with a large number of bogus associative criteria which form a fixed part of the memory. One of the most important questions which must be asked concerning any particular memory is whether that specific selection of bogus associative criteria is complete in the sense that they will cause a data word to be addressed to the same storage location as would have been done by the simultaneous comparison of all data words. This question is treated in detail in Appendix 1; however, the following argument shows that there will always be some selection for which this is true.

The input space over which the data words are defined will always be a finite point set as a consequence of a restriction on each of the descriptors or independent variables that they lie in a finite discrete range. Therefore, the complete memory fill, which is a subset of functions on this point set, is itself finite. It was assumed earlier that the associative criteria were adequate for the unambiguous parallel associative addressing of the data words. If now the selection of fixed associative criteria included every word of the memory fill, then obviously by this assumption each data word would be properly addressed. Equally obviously, if the memory required this number of bogus criteria to operate it would be of no practical interest. The foregoing argument was introduced only to demonstrate that for some finite selection of fixed criteria the data word could be assigned the proper associative address.

Consider a function $f(x_1,x_2, \ldots, x_n)$ defined over n variables, where each of the variables is restricted to some finite discrete set of values, as specifying the environment or source of entries for an associative memory. Then the set of values

$$\{f(x_1, x_2, \ldots, x_n) \; ; x_1, x_2, \ldots x_n\}$$

consisting of the complete specification of a point in the input space and the value assumed by the function at that point is a data word. The second half of the bracketed expression

$$\{\ldots ; x_1, x_2, \ldots, x_n\}$$

in which the x_i elements are treated as the independent variables of the input space is the associative criterion.

Each of the x_i above is restricted to a discrete set of values. Let k_i be the number of increments in the range of x_i. It is an extremely useful convention for the purposes of associative addressing if we think of the points in the input space as represented by a linear array of symbols

$$(x_{11}, x_{12} \ldots x_{1k_1})(x_{21}, x_{22} \ldots x_{2k_2}) \ldots (x_{n1}, x_{n2} \ldots x_{nk_n}) \qquad (1)$$

where each of the parentheses represents all of the values in the discrete range of the corresponding variable. Every point in the input space will be represented as a vector of this form in which a single 1 appears in each parenthesis, with all other entries in that parenthesis being 0. Each of these vectors has N

binary elements, and hence is a vertex of the N-dimensional unit side hypercube situated in the vertex of the first octant in E_N space, where

$$N = \sum_{i=1}^{n} k_i \qquad (2)$$

In the balance of this paper we shall refer to this as the input or S space.

Consider two points in S space represented by their linear vectors (in the same form as (1)); X, X′. Ignoring for the moment the value assumed by the function at these points, the first question, and certainly the most natural one, is "How close are X and X′ to each other?" In order to discuss nearness it is necessary to introduce a metric. The most natural metric for the points in S space as defined by (1) is

$$d^2 = n - X' \cdot X^T \qquad (3)$$

which has the simple interpretation of testing whether the variables are identical in each partition, and of giving as the metric the number of partitions in which X and X′ differ. The parallel associative classification of a memory fill is based on the computation of all such distances between entries.

The points in the original input space, n-dimensional, were mapped onto a select set of the vertices of the unit hypercube in E_N with the restriction that the Hamming weight of these points be precisely n. Similarly the bogus associative criteria, or reference points in S space, will also be chosen to be vertices of this same hypercube, without, however, the restriction to Hamming weights of n. It is most convenient to consider these vertices to have been chosen randomly so that the reference points are uniformly distributed on the hypercube. This definition gives rise to a difficulty which did not exist in the computation of the distance between a pair of associative criteria. There, since the vectors, X, were restricted to have a single 1 in each partition, the distance was either 1, within a single partition, or 0 depending on whether the value of that variable was the same in the two associative criteria, or dissimilar, respectively. Now there is the possibility that a single partition could contain all 1's, for an example, in which case the distance d would be 0 in that partition irrespective of which associative criterion it was compared to. This weakness of the metric d in the enlarged space, i.e., the union of the permissible input S-space and the space made up of the reference points, does not prevent the model from working, as the analysis of Appendix 1 proves, but it does indicate that only a very small portion of the total information available is utilized in forming the associative address. We will still define distance in this enlarged space by the metric d and ignore the degeneracies which can occur.

The vertices of the hypercube chosen as reference points are most conveniently treated as vectors since d^2 is formed by an inner product, and hence we will refer to any individual reference point in the extended S space as an association vector A_i. If there are R association vectors, then they may be best dealt with as an N x R binary association matrix A. The product of an associative criterion, considered as an N element row vector, and A, is a row vector with the R elements being the metric d^2 less n in each case. This vector is designated as the S vector with elements s_i.

$$S = X \cdot A \qquad (4)$$

In Appendix 1 it is shown that for R sufficiently large it is possible to recover d^2 from S and S′, and that therefore the metric is complete over the point set of R-tuples S.

We will now introduce a nonlinear threshold operation on the elements of the vector S. A new (1 x R) row vector α, with elements α_i, will be derived from the S vector by the following procedure:

$$\alpha_i = \begin{cases} 1 & \text{if } s_i \geq \delta \\ 0 & \text{if } s_i < \delta \end{cases}$$

Thus we have generated from the product X · A a binary (1 x R) row vector which has a 1 in the i-th position if the input X is not more than n—δ distant from the reference point A_i. Again it is worthwhile emphasizing that for R sufficiently large the vectors α and α specifying the distance of X and X′ from the R association vectors $A_1, A_2, \ldots A_R$, are at the same time adequate to define the distance d^2 between X and X′. This is true in spite of the weak definition of the metric in the extended space as is shown in Appendix 1. This α vector is the desired associative address.

It is possible to give a simple geometrical interpretation of the associative address. As was noted above the extended S space is the set of vertices of the unit hypercube in E_N. A hypersphere of radius $\sqrt{\frac{N}{2}}$ can be circumscribed about every such hypercube and a central projection of the hypercube onto the sphere can be made which leaves the vertices unchanged. The most important point for our geometric discussion though is that the circle drawn through all vertices of the same Hamming weight of the hypercube on the surface of the hypersphere will be equidistant from the poles of the sphere at (00 . . . 0) and (11 . . . 1). Obviously every point which can exist in the extended S-space lies on this hypersphere. We can now map these hyperspheres on an ordinary sphere in 3-space in which case there will be N—1 circles around the sphere corresponding to Hamming weights of N—1, N—2, . . . 2, 1. Thus the simple projection of the 3-variable hypercube on the sphere yields a representation as shown in Figure 1. Similarly for higher dimensional projections we get N—1 circles, each with $\binom{N}{w}$ points on it of Hamming weight, w.

The points in S space which can be X vectors as given by (1) all have a Hamming weight of n; however, not every point with a weight of n is a permissible X-vector. There are N_H points with a Hamming weight n and only T permissible X vectors.

$$N_H = \left[\sum_{i=1}^{n} k_i \right]$$

$$T = \prod_{i=1}^{n} k_i$$

In general disregarding the points of S space which have a Hamming weight different from n, we have Figure 2 as the projection of the hypercube on the sphere, where the symbol o represents permissible X vectors and the symbol · represents other points which have a Hamming weight of n.

Obviously the distance, d, from any such point, X, to a set of vertices of the hypercube, A_i, could be computed. This is precisely the

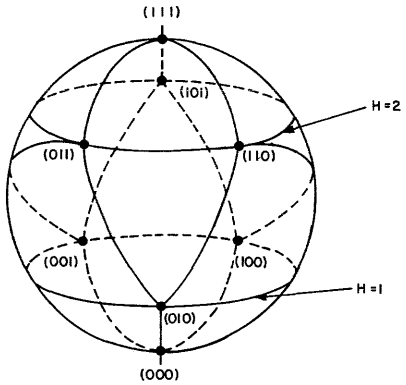


Figure 1.

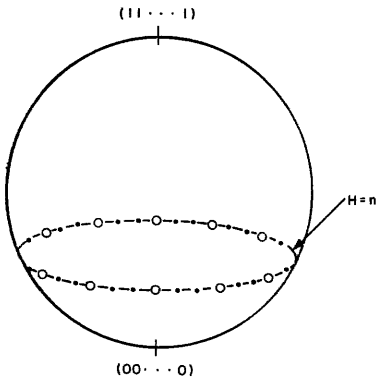


Figure 2.

S-vector discussed above, i.e., Figure 3. The nonlinear thresholding operation consisted of testing each such distance against a reference δ to see if it was greater than or equal to the reference.

$$\alpha_i = \begin{cases} 1 & \text{if } s_i \geq \delta \\ 0 & \text{if } s_i < \delta \end{cases}$$

Recalling the way in which s was formed, i.e., $s_i = X \cdot A_i$, we see that s is largest for the closest reference points. Thus the threshold operation can be geometrically represented by constructing about X as a center a hypersphere of radius δ and determining which of the reference points lie inside or on this sphere. The α_i corresponding to these points will be assigned a value of 1 by the threshold test. The intersection of the hypersphere about X with the hypersphere on which X occurs is a hypersphere of dimension (n—1) as is shown in the following projection (Figure 4) on a sphere in 3-space.

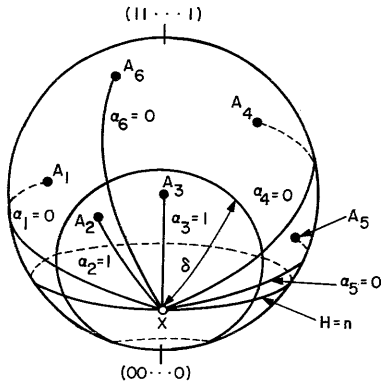


Figure 3.

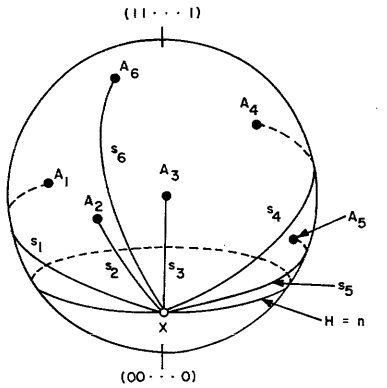


Figure 4.

This completes the geometrical description of the mapping from the N dimensional extended S space to the R dimensional memory space. The resulting set of binary valued α metrics is the associative address.

DISTRIBUTED STORAGE

In the foregoing analysis of a possible associative addressing technique we specifically restricted attention to that fraction of the data word which specified the location of the data point in S-space. This base vector, X, was shown to preserve its metric relationship to other base vectors under a linear transformation followed by a threshold operation which transformed X into a new vector, α. It is this α which we propose to use as an associative address in storing or retrieving data from an associative memory. The most important qualitative point in this procedure is that the transformation maps points from the N-dimensional S-space into points in an R-dimensional space. This tremendous increase in the number of points in space is the basis for the storage of the balance of the data word.

In general, if we consider an R-dimensional space we can position an (R—1) dimensional hyperplane through this space so that the perpendicular distances to R arbitrary points (which are assumed to constitute a basis for the space) from the plane may be arbitrarily specified. The restriction that the points must constitute a basis is equivalent to requiring linear independence of the base vectors from the origin of the system to the points. It is this latter statement which is the essential restriction for the weaker case in which fewer than R points are being fitted to a hyperplane in R-dimensional space.

Let the equation of the hyperplane be given in the form

$$XQ = d$$

where the X and Q are nonzero, R-tuples. Then the equivalent normal form is[18]

$$XL = p$$

where

$$\ell_i = \frac{Q_i}{||Q||}$$

are the cosines of the direction angles of the hyperplane and p is the length of the normal from the origin of the R-space to the hyperplane. This normal, which is of course unique, may be most easily expressed as an R-tuple, with one free parameter, ρ.

$$N = \rho L$$

The ℓ_i are now the direction cosines of the normal. With the notions just developed it is now an easy matter to express the distance d_i of an arbitrary point $\alpha_{(i)}$ to the hyperplane.

$$d_i = p - \alpha_{(i)} \cdot L$$

But this is a linear system, in general nonhomogeneous, of the form

$$\alpha \cdot L = G$$

where G is the R-tuple of elements $(p - d_i)$. Since a solution of such a system for arbitrary G is possible if and only if the elements of the

linear system $\alpha \cdot L$ are linearly independent, we arrive at the earlier restriction that the vectors to the points, $\alpha_{(i)}$, must be linearly independent and therefore, in the case in which there are R-points, must constitute a basis for the space.

As was shown earlier in the description of the formation of S-space and of the procedure by which points in it are mapped into points in M-space, these image points are restricted to be vertices of the R-dimensional binary-valued hypercube positioned in the vertex of the first octant of the space. Actually the restrictions were much tighter than this; however, it suffices to consider any of the vertices of the hypercube in the following development. First we have just shown that a hyperplane can be found such that it lies at a specified, but arbitrary, distance from each of R linearly independent points in R-space, as shown in the following example in 3-space (Figure 5).

Instead of considering a hyperplane at an arbitrary distance p from the origin, which lies at arbitrary but specified distances from the $\alpha_{(i)}$, let us consider a hyperplane through the origin which lies at arbitrary *relative* distances from the $\alpha_{(i)}$. In other words find Q such that

$$\frac{\alpha_{(i)} \cdot Q}{\alpha_{(j)} \cdot Q} = \frac{d_i}{d_j}$$

Recalling that the $\alpha_{(i)}$ are vertices of the R-dimensional hypercube, we see that this is satisfied by letting

$$d_i = \frac{\alpha_{(i)} \cdot Q}{||Q||}$$

which is the quantity we propose to use in storing f(X); i.e., let

$$f(X_{(i)}) = d_i \, ||Q|| = \alpha_{(i)} \cdot Q \qquad (5)$$

We shall hereafter refer to the R-tuple Q as the hyperplane with the understanding that what is actually meant is that Q is the coefficient array of

$$XQ = 0$$

which is a linear equation in R unknowns and hence is the equation of the hyperplane.

It is now possible to construct, in the instance where R linearly independent points are being fitted, a single plane through the origin with the required relative distances, say g_i. Figure 6 shows this for the example used earlier from 3-space.

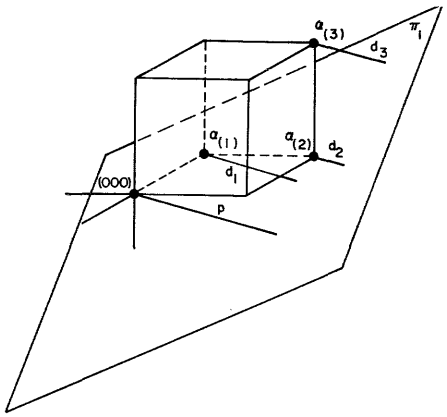


Figure 5.

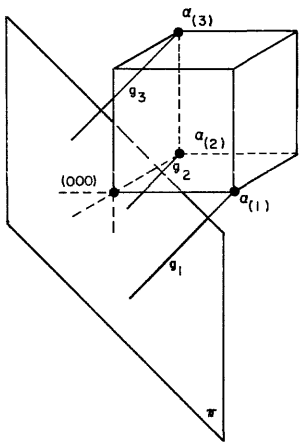


Figure 6.

If fewer than R-points are to be "stored" using this technique, then there are correspondingly infinitely many solutions for each free parameter so introduced. The essential point though is expressed by equation (5), namely that the function to be stored at a point $\alpha_{(i)}$ is given by the normal distance to the hyperplane Q times the norm of Q, i.e., $\alpha_{(i)} \cdot Q$.

The distributed memory is achieved in the R-tuple representation of the solution hyperplane since each element q_i will in general be involved in the computation of the distance to many image points, and any individual distance is dependent on many q_i.

THE ALGORITHM FOR THE SOLUTION OF THE ASSOCIATED LINEAR SYSTEMS

Each data word as it is entered into the memory generates an element of a linear system. This system of T equations in R unknowns could, if it were available at one time, be treated by several conventional methods. Unfortunately the data points are presented serially so that only a single equation is available to the associative memory at any particular time, and we have presupposed no detailed memory for the elements of the linear system. Thus, any solution will have to be generated by a sequential operation on each equation. There is only one technique in use for solving linear systems which has this essential property, i.e., sequential manipulation of one equation at a time. This is the "relaxation method" devised by Gauss and revived and popularized by R. V. Southwell and his colleagues.[19,20] Basically, this procedure depends on the successive minimization of a set of "residuals" associated with the family of linear equations being solved. Let the linear system be of the form

$$\sum_{i=1}^{n} \alpha_{ij} Q_i = f(X_{(i)}) \qquad (6)$$

then the residuals, R_i, are defined to be

$$f(X_{(i)}) - \sum_{i=1}^{n} \alpha_{ij} Q_i' = R_i \qquad (7)$$

where the primed values of Q_i indicate some estimate of the actual vector Q. The most commonly employed algorithm used in relaxation of the family given by (6) depends on computing all of the residuals, $R_1, R_2 \ldots R_T$, for some assumed value of Q. The largest of these residuals is then chosen and a change of the elements in the assumed Q is made so that this residual is caused to become zero (or near zero) and a new family of residuals is computed for this new value of Q'. This procedure is repeated until all residuals are within a tolerable bound of zero, at which time the value of Q' is considered to be the approximate solution for Q. The above description of the relaxation method is based on the most commonly used algorithm, but not the only one used! Other techniques depend on choosing the residual which requires the greatest change in Q' for its liquidation, over-relaxing so that the algebraic sign of the residual being liquidated is changed, and under-relaxing, in which only a part of the residual is liquidated. We do not intend to discuss relaxation methods, as such, and the foregoing was intended only to show the relationship of the algorithm used in the present model to an existing procedure for the solution of a system of linear equations.

In each of the algorithms mentioned above for use with relaxation techniques, the entire system of linear equations was assumed to be available at one time so that all of the residuals could be computed and a decision made, based on the values of these residuals, as to the next step in the process. In the model of the associative memory under discussion only one equation is available at a time, so that its residual can be either partially or totally liquidated, without, however, knowing the effect of this operation on the other residuals.

From equation (7) we may write the linear system, in residual form, as

$$f(X_{(j)}) - \alpha_{(j)} \cdot Q' = R_j \qquad (8\text{-a})$$

where $X_{(j)}$ is the j-th point chosen in S-space (not a component of X) and $\alpha_{(j)}$ is the image of $X_{(j)}$ under A in M-space (again not a component of α). If R_j is to be liquidated, and if no information is available concerning the relative value of a variable (i.e., the effect of a particular variable on all other residuals) which is certainly true in the restricted system under consideration here, and if further all of the coefficients are identical (1 in this case), then the readjustment of Q' to a new value may be best accomplished by dividing the correction equally among all components. What we have said is that we have no information on which to base a selective adjustment of the variables in Q', and hence we will impartially adjust all of them by the same amount. The computational algorithm takes the form:

$$\left.\begin{aligned} \Delta &= k \frac{R_j}{||\alpha||^2} \\ Q'' &= Q' + \Delta \alpha^T \end{aligned}\right\} \qquad (8\text{-b, c})$$

where the primes indicate successive approximations of Q and $0 < k < 1$. k is most often chosen to be 1 so that the residual at each stage is totally liquidated.

With the new estimate for the vector Q obtained from the iterative cycle (8-a,b,c), a new residual will be computed using the next set of coefficients, α_j, when they are generated. Obviously the second serious question concerning this particular model of an associative memory is, "Under what conditions does the iterative procedure converge?" The convergence is analyzed in detail in Appendix 2; however, it suffices for the present purposes to summarize the results obtained there as saying that if the system has a solution, then the procedure will converge to it, and that if the system has no solution it converges to a "best" approximation in a sense which is discussed in Appendix 2.

This completes the description of the associative memory model. We will now discuss the behavior of such a memory simulated on a CDC-1604 computer.

EXPERIMENTAL INVESTIGATIONS

The model for an associative memory developed in the foregoing analysis leaves two principal questions to be answered by experimentation. The first concerns the behavior of the iterative procedure when used to solve linear systems; the second concerns the behavior of the memory, i.e., its convergence during the fill operation and the accuracy of its estimation of previously unfilled points in S-space.

In the first experiments no associative memory will be used; rather a linear system such as might be produced by an associative memory will be contrived to simulate the α vectors and demonstrate the solution behavior of the algorithm.

Experiment No. 1

In this experiment a set of forty linear equations (with binary coefficients) in forty unknowns, and a half set of these equations were solved. It was desirable that these equations all have precisely the same Hamming weight to make them more nearly like the α vectors generated by the associative memory. To accomplish this the following scheme was employed. Each of the possible distinct combinations of two 1's and three 0's were assigned a code number: 0, 1, . . . , 9. A random number generator was used to generate 320 random digits, each block of eight digits corresponding to a single linear element, and the appropriate combination was substituted for the decimal code digits to yield the system of forty random binary coefficients and forty linear elements. The first five elements are given in octal form below to indicate the general character of the linear system.

1 5 1 2 1 4 4 1 6 1 1 3 2 0
0 6 4 5 4 1 4 2 4 3 1 3 0 4
1 2 4 4 3 5 1 2 3 0 0 7 2 0
0 5 1 0 5 4 5 4 2 1 4 5 1 0
1 1 0 5 1 5 0 5 1 1 3 0 1 4

The statistically predicted overlap for such a system is 16%. This compares very well with the actual overlap of 15.8%. We thus have a system with a precisely known Hamming weight, 16 or a density of 40%, and a nearly statistically perfect overlap. One point still has to be insured—the compatibility of the entire linear system. To insure this we assumed a $\hat{Q}$ and computed the functional values to be $\alpha_{(j)} \cdot \hat{Q}$, thus guaranteeing the linear system to be a compatible one, even if the α's failed to be linearly independent (as they did). The $\hat{Q}$ was chosen to be

$$\hat{Q} = (\hat{q}_1, \hat{q}_2, \ldots, \hat{q}_{40})$$

where

$$\hat{q}_i = i; \quad i = 1, 2, \ldots 40$$

The function f computed in this manner has a mean of

$$\bar{f} = \frac{n(n+1)}{2} p = 328$$

and a standard deviation of

$$\sigma = \frac{n+1}{2} \sqrt{np(1-p)} = 63.5$$

This system of forty compatible equations in forty unknowns is the basis for the first portion of this experiment.

The following indicator of convergence was defined:

$$E = \frac{1}{T} \sum_{i=1}^{n} \frac{|R_{(i)}|}{|\alpha_{(i)} \cdot \hat{Q}|}$$

defined over a cycle. E is in some sense a mean

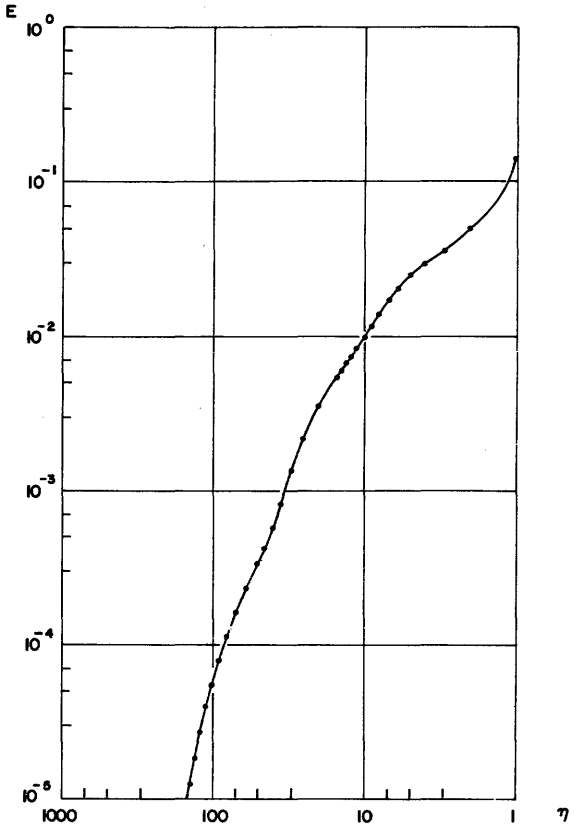


Figure 7. Convergence of a 40 x 40 Linear System.

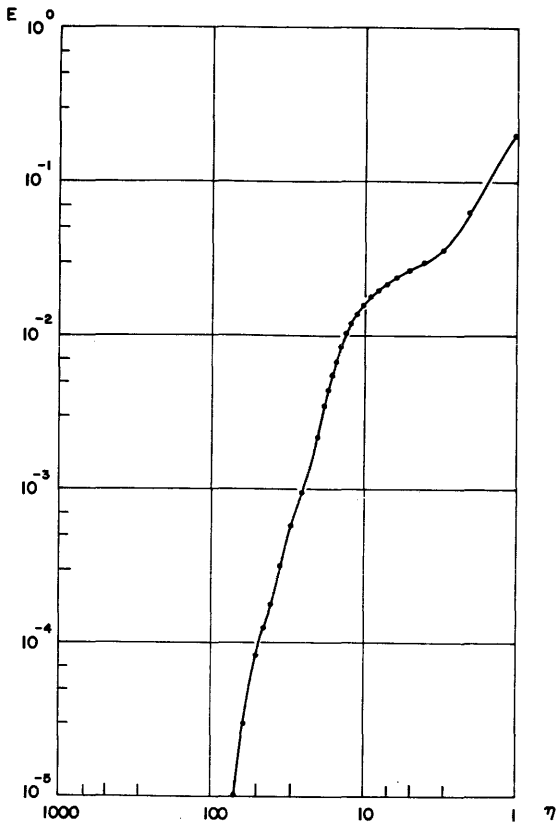


Figure 8. Convergence of a 20 x 40 Linear System.

absolute percentage of error in estimating $f_{(i)} = \alpha_{(i)}$. $\hat{Q}$, and consequently is a useful indicator of convergence. It is important that f be well removed from 0, and that the σ be small enough to not produce large fluctuations in the percentages. The generating scheme described above was chosen to satisfy all of these requirements.

The convergence for the 40 x 40 system is shown in Figure 7. This plot also shows one of the empirically discovered features of this algorithm; namely, that a plot of the logarithm of the mean absolute percentage error versus the logarithm of the cycle number is approximately a straight line. This is qualitatively true of all the systems, which have been solved to date.

The second portion of this experiment consisted of solving the first twenty elements of the preceding linear system, i.e., a linear system of twenty equations in forty unknowns. E is plotted in Figure 8.

Experiment No. 2

The objectives of this experiment are basically the same as those which prompted experiment No. 1; however, there are two differences:

1. The linear system has been increased from 40 x 40 to 100 x 100.
2. The coefficients have been directly assigned by the random number generator so that the Hamming weight is only statistically defined.

The α elements are generated by a random number generator, with the probability of having a 1 in any single position being 0.2. The mean Hamming weight of the α vectors therefore is

$$\overline{H}_a = pn = 20$$

with a standard deviation of

$$\sigma_a = \sqrt{\overline{H}_a(1-p)} = 4.$$

After the α elements were generated by the above scheme, the functional values were as-

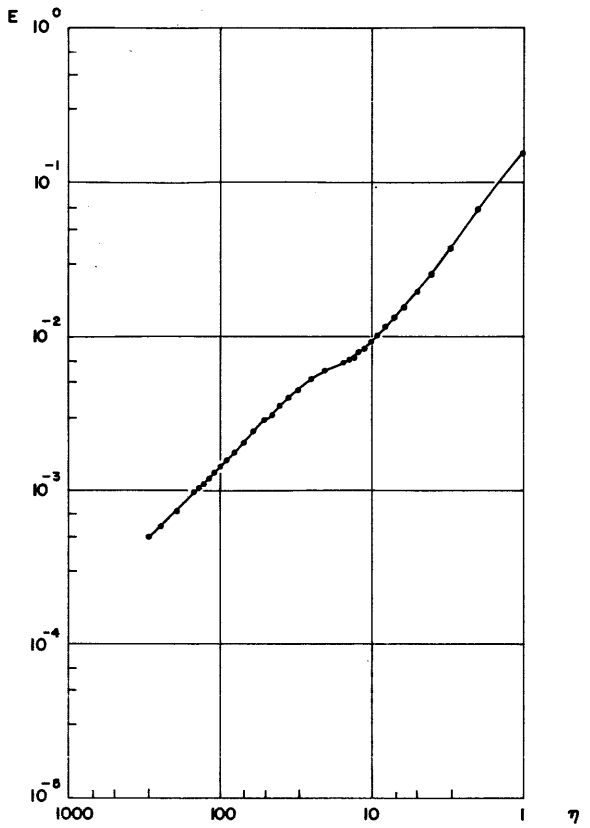


Figure 9. Convergence of a 100 x 100 Linear System.

signed using the same procedure described in Experiment No. 1. The mean functional value then is given by

$$\bar{f} = \frac{n(n+1)}{2} p = 1010$$

with a standard deviation of

$$\delta = \frac{n+1}{2} \sqrt{np(1-p)} = 202$$

In this experiment only the cyclical sequence for relaxation was used. Figure 9 shows a plot of the convergence of the system of one hundred equations in one hundred unknowns as a function of the number of cycles through which the system has been relaxed.

The linear system investigated in this experiment corresponds more closely to the systems which will be generated by the probabilistic associative memory, whereas the system of Experiment No. 1 corresponds more closely to those generated by a complete binary matrix. In the latter case, the Hamming weight of all α elements is exactly the same, while in the probabilistic case the Hamming weight is only statistically specified. One would expect the behavior exhibited in this case to extrapolate directly to larger linear systems and thence to the linear systems generated by the associative memory. The next experiment shows this indeed to be the case.

Experiment No. 3

This last experiment (to be described in this paper) on linear systems differs from the preceding experiment only in the size of the linear system considered—250 equations in 1000 unknowns. The generating procedure for the α elements and the method of assigning the functional values to insure compatibility are the same as those already discussed. The essential parameters descriptive of this problem then are:

size	250 x 1000
p	0.2
$\overline{H}_a$	200
σ_a	12.65
$\bar{f}$	100,100
σ	6,331

The convergence behavior of this linear system, when solved using cyclical relaxation, is shown in Figure 10.

The test problems investigated in this series of three experiments were selected to show the essential properties of solutions generated by nonselective relaxation, i.e., relaxation which is nonselective in the sense that the set of residuals is not examined to determine which linear element is to be relaxed, and also in the sense that the coefficients are not considered in selecting an element (or elements) to be relaxed. In addition to illustrating these general behavioral characteristics, the size of the systems being solved was gradually increased until they were comparable in magnitude to those generated by interesting associative memories. With these comments we leave the subject of the nonselective relaxation technique and consider instead actual associative memory studies.

Experiment No. 4

This experiment is concerned with an application of the associative memory model to a

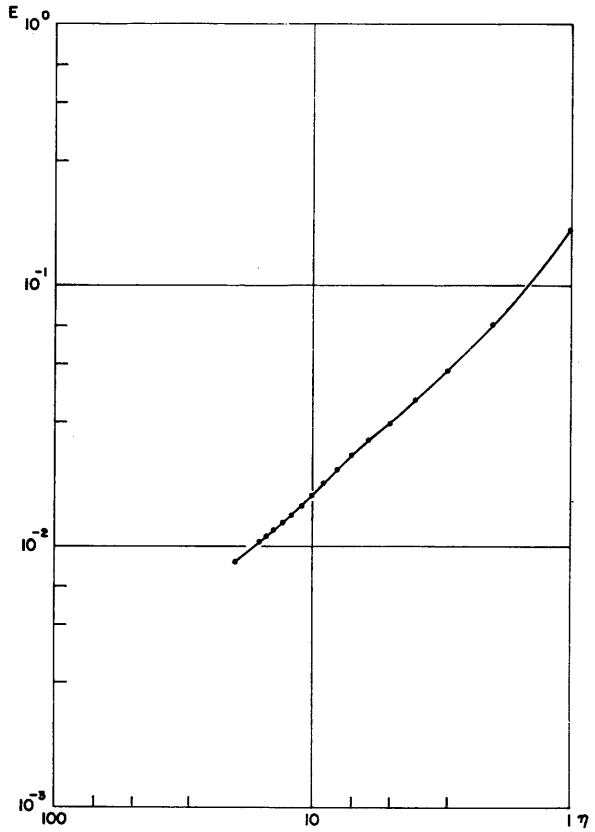


Figure 10. Convergence of a 250 x 1000 Linear System.

problem in pattern recognition, which while simple, is certainly not trivial. The experiment will be described in two stages: first the procedure by which the patterns are generated will be described and analyzed, and then the associative memory which was used and its functioning will be discussed.

It is desirable in order to simplify the analysis of the results that the pattern generation scheme assign approximately half of the possible inputs to the class of patterns and half to the class of nonpatterns. It is also desirable that the procedure have an intuitive appeal as a pattern generator, that is that the patterns should be recognizable by some relatively simple logical test. The following satisfies both of these criteria. Consider a four-element mosaic x_{11}, x_{12}, x_{21}, x_{22}, where each element can assume any one of a discrete range of values, x.

The pattern which is tested for in this experiment consists of a bar, either vertical or horizontal, where both the bar and the background are subject to noise. This is accomplished by the following logical test:

$$f(X) = b_h \oplus b_v$$

where

$$b_h = \begin{cases} 1 & \text{if } \min(x_{11}, x_{12}) > \max(x_{21}, x_{22}) \\ & \text{or } \min(x_{21}, x_{22}) > \max(x_{11}, x_{12}) \\ 0 & \text{otherwise} \end{cases}$$

$$b_v = \begin{cases} 1 & \text{if } \min(x_{11}, x_{12}) > \max(x_{12}, x_{22}) \\ & \text{or } \min(x_{12}, x_{22}) > \max(x_{11}, x_{21}) \\ 0 & \text{otherwise} \end{cases}$$

This scheme has the intuitive appeal, if one interprets the range of possible values of x as a grey scale, of detecting a noisy pattern on a noisy background. The general rule for the recognition of a figure is that

$$\min(F) > \max(B)$$

where F designates the figure elements and B designates the background elements, i.e., the lightest element in the figure is to be darker than the darkest element in the background. Thus a contrast enhancing procedure which converted all elements of a field to black or white depending on whether they were greater than or equal to the minimum value in a potential figure or less respectively would reconstruct a black figure on a white background with no degradation, had the input mosaic pattern actually been in the class of figures. The probability that an arbitrary four-element pattern, with a grey scale of 48 steps, is recognizable as a bar is 0.3197, which provides an adequate density of patterns for the purposes of the present experiment.

The associative addressing matrix used in this experiment had an effective size of 9600 columns, each of which was 192 bits in length. The actual matrix was 1/48-th this size, 192 x 200, but a technique devised by Dr. H. Everett of WSEG makes it possible to use this smaller matrix in such a way as to be equivalent to the larger memory. The matrix is filled with ones and zeroes using a random number generator, with the probability of a one being entered in any bit position being a prechosen value p: in this example $p = 0.2$.

The threshold was selected to be $\delta = 3$. Using these parameters it is possible to define the statistical behavior of the memory in generating the associative addresses. The probability that any particular α element is a one, P , is given by

$$P = \sum_{i=3}^{4} \binom{4}{i} (0.2)^i (0.8)^{4-i} = 0.0256$$

Consequently the mean Hamming weight of the α vectors is

$$\overline{H}_\alpha = 245.76$$

with a standard deviation of

$$\sigma_\alpha = 15.48.$$

This says that the number of α elements which will be operated on in any single relaxation cycle is between 215 and 277 with a 95% likelihood, which is a satisfactory description of the extent of the alteration in Q introduced by relaxation.

The associative memory was shown a sequence of 400 patterns constructed by using a random number generator to assign values to the x_{ij}. Whether a particular mosaic input was a pattern or not was determined by applying the logical test. It is worth noting that these 400 points were drawn from an environment of 5, 308, 416, (48^4), possible mosaic configurations. The scores of the device are given in Table I, as accumulated over cycles of fifty exposures.

TABLE I

Cycle	Number of "patterns" in the cycle	Pattern Errors	Nonpattern Errors
1	30	27	2
2	34	10	12
3	25	4	16
4	31	8	6
5	33	2	11
6	29	3	10
7	26	4	8
8	26	7	4
Totals	234	65	69

The results tabulated in the preceding paragraph summarize quite well the operation of the associative memory as a pattern recognizer. The fact that roughly the same number of patterns and nonpatterns have been misclassified, out of a population of 234 patterns and 166 nonpatterns, is an indication of unbiased performance in the classification of mosaic figures. These results are shown graphically in Figure 11 where the raw scores, accumulated over cycles of fifty exposures, and a smoothed curve are plotted versus cycle number.

Experiment No. 5

This experiment is devoted to an investigation of the behavior of the associative memory when used to store a simple algebraic function originally used by Smith[21] in his studies of a perceptron-like computer model, ADAP II. This particular function was chosen because it embodied several special features which allow a testing of the associative memory functions and because it permits a qualitative comparison of Smith's results and those achieved using the associative memory. The "Smith function" is given by

$$F_s = x_1 x_2 + 2x_3 + 0 \cdot x_4$$

which has a mean of

$$\overline{F}_s = \frac{k_1 k_2}{4} + k_3$$

where k_i is the number of nonzero steps in the discrete range of the i-th variable. For test purposes this function has several advantages. It involves a cross product term which has proven

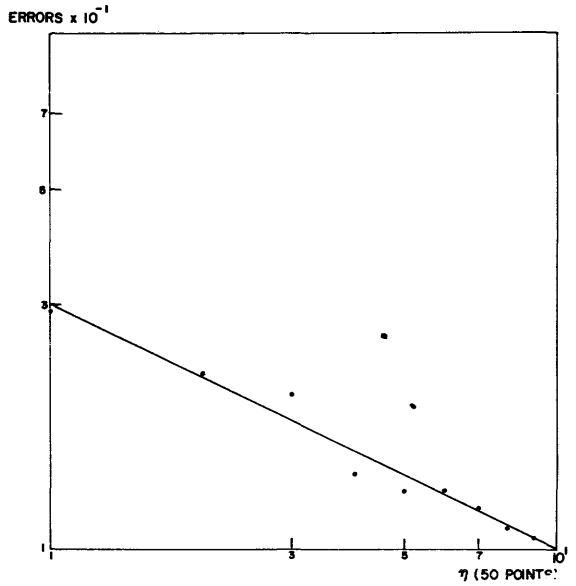


Figure 11. Scores of the Pattern Recognition Program on Cycles of 50 Points Each.

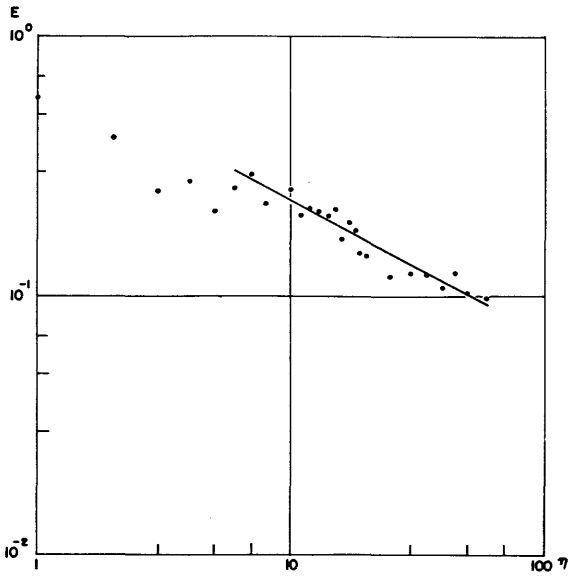


Figure 12. Convergence for F_s with $p = 0.2$; $\delta = 3$.

to be a very difficult function for perceptron-like (single layer perceptrons) devices to learn. It has a term which has a zero coefficient and hence, although this variable enters into the associative addressing, cannot affect the functional value. Finally the basic form is a linear summation of terms, a functional form most easily stored in the quasi-linear associative memory. In all of the tests which make up this experiment $n = 9$, i.e., $x_i = (0, 1, \ldots, 9)$. The matrix is 192 x 9,600 with the Everett scheme for permuting a base matrix being applied to a basic 192 x 200 array.

The essential parameters of the experiment are

$$p = 0.2$$
$$\delta = 3$$

which gives a mean Hamming weight of

$$H_a = 261.12$$

A random number generator was used to assign random values (0 — 9) to x_1, x_2, x_3, and x_4 and the corresponding F_s was calculated using these random variables. The convergence plot for this test is shown in Figure 12. E is defined in a slightly different manner here than before. In the case of the contrived compatible linear systems where E was first introduced, the functional values were tightly clustered about the mean and bounded away from zero so that no difficulty could be encountered in defining E to be the average absolute percentage error. Here the function is very broadly spread, and furthermore not bounded away from zero; therefore the following related definition for E is used in discussing this experiment.

$$E = \frac{1}{D} \sum_{i=1}^{D} \frac{|R_{(i)}|}{\overline{F}_s}$$

where $D = 60$ in this experiment.

The following test is actually the crux of this whole series of experiments since it demonstrates the actual operation of the associative memory. This test is concerned with the memory's performance on the F_s function using the parameters given in the preceding paragraph. The convergence plot for this particular test has already been given in Figure 12; however, the detailed behavior of the memory is lost in a smoothed measure of convergence such as E. Since the primary function of the associative memory is the recall (or estimation) of functional values it is highly desirable to present the memory output in a form where its performance can be conveniently judged.

Two questions which occur naturally when one considers an associative memory are:

1. How will it perform in a new field of data

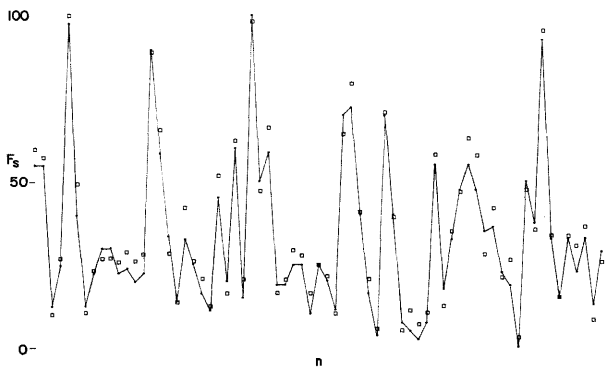


Figure 13. Test of a Filled Memory on Random Points in X Space (□ = predicted, • = actual values).

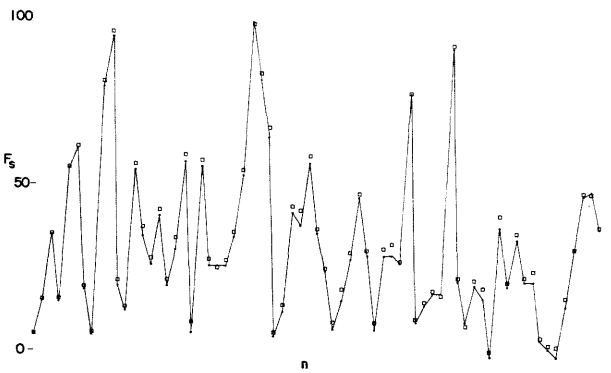


Figure 14. Test of the Survival of Detailed Memory Entries (□ = predicted, • = actual values).

points, if forced to estimate without new data being entered?

2. What is the span of its memory; that is how well will it do if shown the same set of data points with which the memory was originally filled, compared to a new field of data points?

Both of these questions have been investigated. The memory was "filled" by drawing 3000 data points at random as described before, and then by setting the relaxation coefficient to zero, $k = 0$, further changes in the Q vector were prevented. First a sequence of random points, presumably new ones, were used to query the memory. The results of this test are shown in Figure 13. $E = 0.10444$ which indicates that the terminal performance of the associative memory while in the learn cycle, $E = 0.10248$, is probably a measure of its storage accuracy over the entire functional space. The second question actually answers this by showing that the detailed memory of the actual data points entered in the memory has been sacrificed for a type of distributed functional memory for the entire functional space. Figure 14 shows this phenomenon in a very striking fashion. The points immediately preceding the end of the storage period are all queried, with the unexpected result that the memory has a detailed retentivity period of perhaps 500 points, or half a cycle, and that beyond this the individual data points have been lost individually and have been assimilated instead in a collective form as a "smoothed" function.

In conclusion then, the series of tests which made up Experiment No. 5 has demonstrated the behavior of the associative memory on an actual notrivial problem. The data could just as well have been drawn from any number of real-life problem situations; however, it seemed appropriate to discuss a well-behaved analytic function in this paper to permit the reader to readily judge the performance of the memory.

CONCLUSION

In view of the length of some of the arguments which have been presented, it seems especially desirable to give a succinct description of the mathematical model for an associative memory developed in this paper.

A function (commonly a decision function) $f(X)$ is defined over an n-dimensional input

space, in which each of the variables is restricted to assume one of a discrete set, k_i in number, of values. Each of these input points is mapped linearly onto a vertex of an N-dimensional unit hypercube,

$$N = \sum_{i=1}^{n} k_i$$

all of whose vertices make up the so-called S-space. The function $f(X)$ is the desired memory fill.

A metric s_i is defined by

$$S = X \cdot A$$

where A is an (N x R) matrix whose columns are points in S-space. These points, A_i, constitute a set of R fixed reference points in S-space. The metric s_i defines the distance of X from A_i, and was shown to be complete. A much weaker metric, α_i, is defined on S by the logical operation

$$\alpha_i = \begin{cases} 1 & \text{if } s_i \geqq \delta \\ 0 & \text{if } s_i \geqq \delta \end{cases}$$

where δ is a threshold value; $0 \leqq \delta \leqq n$.

It was shown that for some A (i.e., for a sufficiently large number of fixed reference points) there exists at least one hyperplane, Q, in the M space in which α is defined, such that the distance at which the point α, whose pre-image as defined above is X, lies from Q is the function $f(X)$ multiplied by a suitable normalizing constant. Such a hyperplane may be found by the iterative procedure

$$\left.\begin{aligned} R &= f(X) - \alpha \cdot Q' \\ \Delta &= \frac{kR}{||\alpha||^2} \\ Q'' &= Q' + \Delta\, \alpha^T \end{aligned}\right\} \qquad (8)$$

which was proven in Appendix 2 to converge.

Thus we may finally conclude that the output of this particular model of an associative memory (at a time where Q' is the estimate of Q) upon being interrogated with the address (associative criterion) X is

$$F(X) = \alpha(X) \cdot Q'$$

APPENDIX 1. COMPLETENESS OF THE d^2 METRIC

In the parallel associative addressing of data, the distance d between words is computed and the entries are organized accordingly. d^2 was defined as a function of the N-tuples representing the points in S-space and one must show that the same distance can be calculated for any of the proposed image spaces. In this appendix we shall prove that d^2 is the same for X and X', S and S' and α and α'.

Following our earlier usage the vectors in S-space will be assumed to have N elements grouped in n partitions or variable ranges in each of which a single one occurs. Now let A be the complete N x 2^N binary matrix in which every possible N bit binary word appears as a column of the matrix. The S matrix is then a (1 x 2^N) row matrix formed by multiplying A by X. If we consider two vectors, say X and X', in S-space the distance between them is found from (3) to be

$$d^2 = n - X'X^T \qquad (3)$$

The first step in demonstrating that the weak metric on S is a complete one is to show that d^2 as given by (3) may be computed from S and S' and non-specific knowledge about S-space:

$$XA = S \quad \text{and} \quad X'A = S'$$

form

$$X'A(XA)^T = S'S^T$$

or

$$X'AA^TX^T = S'S^T \qquad (9)$$

The unusual properties of the complete binary matrix make it possible to greatly simplify the left hand side of (9) through a manipulation of AA^T

$$AA^T = (a_{ij}) \begin{cases} a_{ij} = 2^{N-2} & \text{if } i \neq j \\ a_{ij} = 2^{N-1} & \text{if } i = j \end{cases}$$

irrespective of any ordering of the A_i in A. Thus AA^T can be decomposed to the sum of two particularly simple matrices, i.e.,

$$AA^T = 2^{N-2} \begin{bmatrix} 1 & \cdots & 1 \\ \cdot & & \cdot \\ \cdot & & \cdot \\ \cdot & & \cdot \\ 1 & \cdots & 1 \end{bmatrix} + 2^{N-2} I$$

This formula may now be re-introduced into (9) to give the following very simple form

$$2^{N-2} X' \begin{bmatrix} 1 & \cdots & 1 \\ \cdot & & \cdot \\ \cdot & & \cdot \\ \cdot & & \cdot \\ 1 & \cdots & 1 \end{bmatrix} X^T + 2^{N-2} X'X^T = S'S^T \qquad (10)$$

If one notes that the effect of the matrix operation in the left hand member is merely to tally twice the nonzero elements of X′ and X^T and that this number is n, then (10) can be reduced and the resulting equation solved for $X'X^T$

$$X'X^T = \frac{1}{2^{N-2}} \{ S'S^T - n^2 2^{N-2} \} \quad (11)$$

Finally the distance between X and X′ is found to be

$$d^2 = n - \frac{1}{2^{N-2}} \{ S'S^T - n^2 2^{N-2} \} \quad (12)$$

where the right hand term is expressed only in terms of S, S′ and parameters descriptive of the general characteristics (n and N) of S-space as was desired. This proves that for some R, in this case $R = 2^N$, the metric defined on S is a complete one, that is to say, that the location of a point in S-space is unambiguously specified by the S vector.

First consider the form of the S matrix, $S = X \cdot A$. It will have integer elements with all integer values 0 thru n being represented. This can be decomposed into the following useful form

$$S = \sum_{i=1}^{n} S(i) \quad (13)$$

where each of the S(i) is the matrix whose elements are either i or 0. This representation of S allows one to represent α analytically in the following form.

$$\alpha = \sum_{i=\delta}^{n} \frac{1}{i} S(i) \quad (14)$$

where δ is the threshold value chosen for the logical operation in (6).

The basic problem is: given α and α', is it possible to compute the distance between X and X′, or in other words is the weak metric α complete in the extended space? Note that the value of the threshold δ against which α is computed is left unspecified, and hence can be considered as one of the parameters of the problem. Assume δ to be n; in other words the association vector must have a 1 in precisely every code position in which the X vector has a 1 if the corresponding α element is to be nonzero. As we have noted before, this result is not affected by the presence of extra 1's in the association vector. α is now a single term and may be written as

$$\alpha = \frac{1}{n} S(n) \quad (14\text{-a})$$

It is now possible to compute the number of nonzero elements in α rather simply. If there are n partitions with k_i discrete values possible in the i-th partition, and if the association matrix is the same complete binary matrix discussed before, the number of elements whose value is n in S is

$$\phi_0(n) = \prod_{i=1}^{n} 2^{k_i - 1} \quad (15)$$

If we consider another vector with (n — 1) of the partitions identical, with a single element different in a partition possessing k_j elements, then the number of $\alpha_i = n$ in the same relative positions of the two α vectors is

$$\phi_1(n) = \prod_{\substack{i=1 \\ i \neq j}}^{n} 2^{k_i - 1} \times 2^{k_j - 2} \quad (16)$$

This argument is simply extended to the case where the two X vectors differ in an arbitrary number, $p \leq n$, of partitions

$$\phi_p(n) = \prod_{\substack{i=1 \\ i \neq j}}^{n} 2^{k_i - 1} \prod_{j=j(1)}^{j(p)} 2^{k_j - 2} \quad (17)$$

It is possible to express X as a function of α as defined by (14-a), $\phi_0(n)$ as given by (15) and A, or more precisely A^T. We shall now prove the following theorem.

$$X = \left[\frac{\phi_0(n)}{2}\right]^{-1} \left\{ \alpha A^T - \frac{\phi_0(n)}{2} (1 \ldots 1) \right\} \quad (18)$$

First consider the term αA^T. α_i is a one, if and only if, s_i is equal to n, which is to say that the binary cycles of the rows of A, assumed to be ordered as shown in the previous example for this argument (which correspond to a 1 in the elements of the X matrix) must all be positive, or equal to one in this case. Thus, the operation of matrix multiplication followed by the logical decision (6) is exactly analogous to a double logical extract operation. If we now reverse

this operation and extra α against A^T, i.e., the logical operation

$$\sum_{i=1}^{n} \alpha_i A_{ij} = \beta_j \qquad (19)$$

The rows of A which contributed to making s_i pass the threshold test will now be "in phase" with the variations of α_i; that is they will be positive in all cases when α_i is positive. All other rows will assume all possible binary arrangements while these are fixed and consequently will be positive and zero equally often. Thus the "in phase" rows will contribute a 1 for each 1 in α, or what is the same thing, if R is understood to represent a general "in phase" row of A and R^T is the corresponding column of A^T

$$\alpha \cdot R^T = \sum_{i=1}^{n} \alpha_i = \phi_0(n) \qquad (20)$$

and similarly if r is a general "out of phase" row of A, and r^T is the corresponding column of A^T

$$\alpha r^T = 1/2 \sum_{i=1}^{n} \alpha_i = \frac{\phi_0(n)}{2} \qquad (21)$$

But if one recalls the definition by which the terms "in phase" and "out of phase" were introduced, namely, that an in phase row of A is one which corresponds in position to nonzero element of X and conversely for the out of phase rows, we see that the vector X is derivable from αA^T. αA^T is therefore a (1 x N) row vector which has $\frac{\phi_0(n)}{2}$ in every position in which X has a zero, and $\phi_0(n)$ in every position in which X has a one. Equation (18) is thus obtained by a trivial algebraic manipulation upon αA^T.

The original question which prompted this discussion can now be answered in the affirmative—for an appropriate choice of the threshold, δ, and for R sufficiently large (at least for the case $R = 2^N$ as we have demonstrated) X can be expressed as a function of its distance from each of the association vectors in terms of the weak metric α. Certainly if X and X′ can be expressed in this manner, the distance between them, d^2, can also be expressed in terms of the same variable since d^2 is defined on X and X′. This is incidentally a much stronger result than the one which we set out to prove which only required that we be able to find d^2, not that we be able to solve uniquely for X and X′. The weaker case is interestingly enough much harder to prove; however, it is also much more valuable in the application of the construction to modeling real associative memories.

Having completed all of the preliminary formulations required to compute d^2 for any vector pair, X and X′, in S-space it is an interesting exercise to compute the formula for d^2 itself.

$$d^2 = n - \left\{\frac{2\alpha' A^T}{\phi_0(n)} - (1 \ldots 1)\right\} \left\{\frac{2\alpha A^T}{\phi_0(n)} - (1 \ldots 1)\right\}^T$$

This may be expanded into the following form which is more amenable to simplification and computation.

$$d^2 = n - N + \frac{4}{\phi_0(n)} \alpha' A^T A \alpha^T - \frac{2}{\phi_0(n)} (1 \ldots 1)\, A\alpha^T - \frac{2}{\phi_0^2(n)} \alpha' A^T \begin{bmatrix} 1 \\ \cdot \\ \cdot \\ \cdot \\ 1 \end{bmatrix} \qquad (22)$$

The two matrix expressions

$$(1 \ldots 1) A\alpha^T \text{ and } \alpha' A^T \begin{bmatrix} 1 \\ \cdot \\ \cdot \\ \cdot \\ 1 \end{bmatrix}$$

may be reduced to their equivalent scalars. First by the argument of the previous section $\alpha'\, A^T$ is a (1 x N) row matrix which has the element $\frac{\phi_0(n)}{2}$ in every position in which X had a zero and the element $\phi_0(n)$ in every position in which X had a one. A similar statement applies to the column matrix $A\alpha^T$. The final matrix operations

$$(1 \ldots 1) X \text{ and } X \begin{bmatrix} 1 \\ \cdot \\ \cdot \\ \cdot \\ 1 \end{bmatrix}$$

simply sum these elements so that one may summarize these results by

$$-\frac{2}{\phi_0(n)}(1\ldots1)A\alpha^T = -\frac{2}{\phi_0(n)}\alpha A^T\begin{bmatrix}1\\ \cdot\\ \cdot\\ \cdot\\ 1\end{bmatrix} = -(N+n) \qquad (23)$$

which makes it possible to express (22) in the following simple form

$$d^2 = (N+3n) - \frac{4}{\phi_0^2(n)}\alpha' A^T A\alpha^T \qquad (24)$$

which may itself be simplified even further by expanding A^TA into a summation of simple matrices.

The results expressed by formulas (12) and (24) show that d^2 is computable from the images of X and X′ into S and S′ or α and α' respectively and that therefore the metric d^2 is complete in all three spaces as was to be shown.

APPENDIX 2. CONVERGENCE PROOF FOR THE ALGORITHM

The iterative procedure, which we have called nonselective relaxation, upon which the memory fill is dependent, must be proven to converge. It is much easier to approach such a proof by reducing the procedure to its geometrical equivalent.

Let the estimate of Q stored in the memory at the time a data word {f(x); X} appears be Q′, then the iterative sequence is:

$$\begin{aligned} R &= f(X) - \alpha \cdot Q' \\ \Delta &= \frac{kR}{||\alpha||^2} \\ Q'' &= Q' + \Delta\alpha^T \end{aligned} \qquad (8)$$

It is convenient to use the duals of α and Q, i.e., instead of the α being points in M space and Q a hyperplane through the origin, we consider the α to be hyperplanes whose normal from the origin was the point α before, and Q becomes a point in S space. Using this convention the sequence (8) has a very simple geometrical interpretation. R is the normal distance of the point Q from the hyperplane defined by α. Q″ then becomes the normal orthogonal projection of Q′ into the hyperplane, as Figure 15 illustrates, π_i is the hyperplane defined by $\alpha_{(i)}$. In the above example it is obvious that Q will converge to the solution point Q; however, a general proof is required for higher dimensional spaces, and for over determined systems.

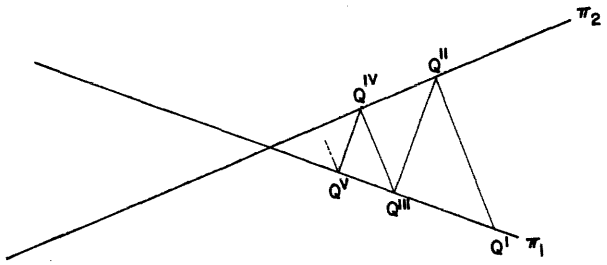


Figure 15.

Let π_i (i = 1, 2, . . . , R) be a family of (n — 1) dimensional hyperplanes defined by the $\alpha_{(i)}$ in E_n, and let the indices designate some arbitrary ordering. We will represent by the symbol $T_{i,i+1}$ the projection of π_i into π_{i+1}, perpendicular to π_{i+1}, from the point at infinity, i.e., a parallel orthogonal projection of π_i onto π_{i+1}. We shall define the projective transformation P to be the product of the R projections T,

$$P = T_{12}T_{23}\ldots T_{R1}$$

which is permissible since the product of arbitrarily many projections is itself a projectivity. From the definition of P it follows that the image of any point in π_1 under P is also in π_1.

Theorem: Let Q be an arbitrary point of π_1, then the limit of P^nQ_0 exists, and furthermore is a fixed point of P, i.e.,

$$\lim_{n\to\infty} P^nQ_0 = Q \qquad Q_0,\ Q \epsilon \pi_1$$

and

$$PQ = Q$$

Preliminary Remarks:

1. There is at least a single fixed point for every projectivity,[22,23] and hence for P in particular.
2. The transformations $T_{i,i+1}$ as defined above have the following property, where

$$||TQ_1^i - TQ_0^i|| = k_i||Q_1^i - Q_0^i|| \qquad (25)$$

where

$$Q_1^i,\ Q_0^i \epsilon \pi_i,\ TQ_1^i,\ TQ_0^i \epsilon \pi_{i+1}$$

and T has been written for $T_{i,i+1}$ since no con-

fusion can result. k_i is a non-negative scalar bounded by the following inequality

$$\cos \Theta_i \leq k_i \leq 1$$

and

$$\cos \Theta_i = \frac{\alpha_i \cdot \alpha_{i+1}}{||\alpha_i|| \quad ||\alpha_{i+1}||}$$

$\frac{||\alpha_i||}{\alpha_i}$ is the unit vector along the normal from the origin to the hyperplane π_i.

k_i cannot be specified more tightly without knowing the orthogonal projection of the line segment $[Q_1^i, Q_0^i]$ on the intersection of π_i and π_{i+1}.

Proof: Since $T_{i,i+1}$ is a linear transformation we may write

$$||T_{j,j+1}Q_1^j - T_{j,j+1}Q_0^j|| - ||T_{j,j+1}T_{j-1,j}Q_1^{j-1} - T_{j,j+1}T_{j-1,j}Q_0^{j-1}|| \quad (26)$$

etc. for j steps in all; where

$$Q_1^j, Q_0^j \in \pi_j \text{ etc.}$$

However, stopping with the step shown in (26) and using (25) we may write

$$||TQ_1^j - TQ_0^j|| = k_j||Q_1^j - Q_0^j|| = k_j||TQ_j^{j-1} - TQ_0^{j-1}||$$

where the T in the left hand norm is $T_{j,j+1}$ and in the right hand norm is $T_{j-1,j}$. Now by repeatedly applying this reduction, j times in all, we obtain

$$||T_{j,j+1}Q_1^j - T_{j,j+1}Q_0^j|| = \prod_{i=1}^{j} k_i||Q_1 - Q_0|| \quad (27)$$

where

$$Q_1, Q_0 \in \pi_1.$$

The R hyperplanes are considered in a cyclical order; therefore on the R + 1-st projection the images will be in π_1, and from (26) and (27) we have:

$$||PQ_1 - PQ_0|| = K||Q_1 - Q_0|| \quad (28)$$

where

$$K = \prod_{i=1}^{R} k_1$$

P is a projectivity of π_1 onto itself, and as such (as noted in Remark 1) has at least one fixed point, or in general some fixed subspace S such that for every

$$\sigma \in S$$
$$P\sigma = \sigma$$

Now choose an arbitrary $Q_0 \in \pi_1$, and let Q_1 be the image of Q_0 under P

$$PQ_0 = Q_1$$

Case 1. If $Q_0 \in S$ then

$$PQ_0 = Q_0$$

and Q_0 is a fixed point of the transformation and the theorem is trivially true.

Case 2. Let $Q_0 \in \pi_1$; $Q_0 \in S$ then

$$Q_0 \neq Q_1$$

and at least one $k_i \neq 1$. $k_j = 1$ is equivalent to saying that the norm of the segment $[Q_1^j, Q_0^j]$ is the same as the norm of the orthogonal projection of $[Q_1^j, Q_0^j]$ on the intersection of π_j and π_{j+1}, say ρ_j. But then for all ρ_j to be 1 would require that the ρ_j be pairwise parallel, i.e., ρ_j parallel to ρ_{j+1}, etc., and $Q_0 = Q_1$ necessarily, in contradiction to the assumption that $Q_0 \neq Q_1$. Therefore at least one $k_1 \neq 1$ and

$$K = \prod_{i=1}^{R} k_i < 1$$

Equation (28) may be expressed as

$$||PQ_j - PQ_{j-1}|| = K||Q_j - Q_{j-1}|| \quad (29)$$

where we need not show superscripts on the Q_j since all points and their images under P are in π_1 alike. Equation (29) may be recursively solved to yield

$$||PQ_j - PQ_{j-1}|| = ||Q_{j+1} - Q_j|| \leq K^j||Q_1 - Q_0|| \quad (30)$$

where

$$||Q_1 - Q_0|| > 0.$$

Now we wish to determine the Cauchy difference for the sequence $\{Q_j\}$; $|Q_{j+m} - Q_j||$. Using (30) repeatedly we obtain

$$||Q_{j+m} - Q_j|| \leq (K^{j+m-1} + K^{j+m-2} + \cdots + k^j||Q_1 - Q_0||$$

after the usual introduction of the missing terms and appropriate regrouping.

$$||Q_{j+m} - Q_j|| \leq K^j\left(\frac{1 - K^m}{1 - K}\right)||Q_1 - Q_0|| \quad (31)$$

but for a suitable choice of j the right hand side of (31) can be made less than any $\epsilon > 0$; therefore the sequence $\{Q_j\}$ converges to a point Q; $Q \in \pi_1$.

We must still prove that Q is indeed a fixed point of P, i.e., $Q \in S$.

$$||PQ - Q|| \leq ||PQ - PQ_j|| + ||PQ_j - Q||$$
$$\leq K||Q - Q_j|| + ||Q_{j+1} - Q||$$

but both of the right hand terms have been shown to go to zero as j increases; therefore $||PQ - Q|| < \epsilon$ for every $\epsilon > 0$ and Q is shown to be a fixed point of P, or

$$PQ = Q \Rightarrow Q \in S$$

Corollary: If the system of hyperplanes defines a determinate system of linear equations with a unique solution point, then this point is the only fixed point under any of the possible projective cycles P, i.e., any reordering of the indices, and the above proof shows that

$$\lim_{n \to \infty} P^n Q' = Q$$

where Q′ is any point in any plane and Q is the unique solution point of the system.

The foregoing argument completes the convergence proof for the algorithm proposed for filling the associative memory.

REFERENCES

1. W. L. McDermid and H. E. Peterson, "A Magnetic Associative Memory System," *IBM Journal* 4, pp. 59-62, 1959.
2. J. R. Kiseda, H. E. Peterson, W. C. Seelbach, and M. Teig, "A Magnetic Associative Memory," *IBM Journal* 5, pp. 106-121, 1961.
3. R. R. Seeber and A. B. Lindquist, "Associative Memory with Ordered Retrieval," *IBM Journal* 6, pp. 126-136, 1962.
4. R. R. Seeber, "Cryogenic Associative Memory," National Conference of the Association for Computing Machinery, August 23, 1960.
5. R. L. Boyel, "A Semantically Associated Memory," pp. 161-169 in *Biological Prototypes and Synthetic Systems* 1, edited by E. E. Bernard and M. R. Kare, Plenum Press, New York, 1962.
6. B. Widrow, "Generalization and Information Storage in Networks of Adaline 'Neurons'," pp. 435-461, in *Self-Organizing Systems,* 1962, edited by M. C. Yovits, *et al.,* Spartan Books, Washington, D. C., 1962.
7. McDermid and Peterson, *op. cit.*
8. Kiseda, *et al., op. cit.*
9. Seeber and Lindquist, *op. cit.*
10. Kiseda, *et al., op. cit.*
11. K. S. Lashley, *Brain Mechanisms and Intelligence:* a quantitative study of injuries to the brain, University of Chicago, Chicago, 1929.
12. F. Rosenblatt, "The Perception—a theory of statistical separability in cognitive systems," Cornell Aeronautical Laboratory Report No. VG-1196-G1, January 1958.
13. F. Rosenblatt, *Principles of Neurodynamics — perceptrons and the theory of brain mechanisms,* Spartan Books, Washington, D. C., 1962.
14. F. Rosenblatt, "Perceptual Generalization over Transformation Groups," pp. 63-100 in *Self-Organizing Systems,* 1960, edited by M. C. Yovits and S. Cameron, Pergamon Press, New York, 1960.
15. B. Widrow, "Adaptive Sampled-Data Systems—A Statistical Theory of Adaption," 1959 WESCON Convention Record, Part 4.
16. B. Widrow and M. E. Hoff, "Adaptive Switching Circuits," Tech. Report No. 1553-1, Stanford Electronics Lab., Stanford, California, June 30, 1960.
17. B. Widrow, "Generalization and Information Storage in Networks of Adaline 'Neurons'," *Self-Organizing Systems,* pp. 435-461, Spartan Books, Washington, D. C., 1962.
18. D. M. Y. Sommerville, *An Introduction to the Geometry of N Dimensions,* Dover Publications, Inc., New York, N.Y., 1958.
19. R. V. Southwell, *Relaxation Methods in Engineering Science—a treatise on approximate computation,* Oxford University Press, Oxford, 1940.

22. HERBERT BUSEMANN and PAUL J. KELLY, *Projective Geometry and Projective Metrics,* Academic Press, Inc., New York, New York, 1953.

23. W. V. D. HODGE and D. PEDOE, *Methods of Algebraic Geometry,* Cambridge University Press, Cambridge, 1947.

20. F. S. SHAW, *An Introduction to Relaxation Methods,* Dover Publications, Inc., New York, 1953.

21. J. W. SMITH, "ADAP II—an adaptive routine for the LARC computer," unpublished report (Navy Management Office) communicated by the author, September 1962.

DESIGN OF AN EXPERIMENTAL MULTIPLE INSTANTANEOUS RESPONSE FILE*

E. L. Younker, C. H. Heckler, Jr., D. P. Masher, and J. M. Yarborough
Stanford Research Institute
Menlo Park, California

SUMMARY

An experimental model of an electronic reference retrieval file in which all file entries are interrogated simultaneously has been designed and constructed. The experimental model is designed to store and search on a file of indexes to 5,000 documents. A document index consists of a decimal accession number and up to eight English word descriptors that are closely related to the contents of the document. The vocabulary required to describe the documents is held in a machine dictionary that has a design capacity of 3,000 words. In the model delivered to the sponsor, Rome Air Development Center, the storage capacity is only partially used. The specification for the delivered model calls for the storage of approximately 1,100 documents that were selected from the ASTIA (now DDC) Technical Abstract Bulletin and of the vocabulary needed to describe them (about 1,000 words). The document indexes and the dictionary words are stored in wiring patterns associated with arrays of linear ferrite magnetic cores.

A search question, consisting of one to eight descriptors in their natural English form, is entered by means of an electric typewriter. During entry of the search question, the dictionary magnetic store is interrogated by the alphabetic code of each search word. If a word is not contained in the dictionary, it is automatically rejected. After all words of the search question have been entered, the document magnetic store is interrogated by the search question in superimposed code form. The comparison between the search word and the document indexes is made for all documents simultaneously and the machine instantaneously determines if any documents in the file include the search question. If there are none, the machine indicates visually that there is no response. If there is at least one, the machine counts the number of responding documents and displays this number. Then it types out the indexes of all responding documents on the same typewriter that was used to ask the question.

INTRODUCTION

Memories that can be searched in parallel and from which stored information is retrieved on the basis of content have received considerable attention for application to retrieval file problems.[1, 2, 3, 4] This paper describes the design of an experimental retrieval file based on the work reported by Goldberg and Green.[3] Since the contents of the semipermanent magnetic memory used in the experimental file can be searched in parallel and multiple responses

* The work described in this paper was supported by Rome Air Development Center under Contract AF 30(602)-2772.

to the search question are permitted, the system is called MIRF—Multiple Instantaneous Response File.[5]

LOGICAL ORGANIZATION OF THE MIRF SYSTEM

The logical organization of the experimental MIRF system is illustrated by Fig. 1. Information pertaining to the document indexes and to the descriptors used in the document indexes is contained in two major units called MIRF units. A MIRF unit is basically a magnetic memory in which information is permanently stored in the wiring associated with the magnetic cores. The Document MIRF is the principal element of the system. It contains for each stored document index the document accession number and the descriptors (in coded form) that describe that document, as well as a superimposed search code that is used in the searching process. The Dictionary MIRF has two functions. During the input phase of operation it translates the alphabetic code of the English word descriptor that is entered from the typewriter into the binary serial number assigned to that English word for use inside the machine. During the output phase, the Dictionary MIRF translates the binary serial number of a word that is obtained during a search into the alphabetically coded form of that word.

After the binary serial number of an input English word has been generated, this binary number is translated by a logical process in the Search Code Generator into a search code that is assigned to the particular English word. The search codes of successive words of a search question are superimposed by adding them together, bit by bit by an inclusive-OR operation. When the search question is complete, the superimposed search code of the question is compared with the superimposed code section of the Document MIRF. Each document index whose search field includes the superimposed code of the search question is said to *respond* to the question. Frequently more than one document will respond. By a logical process for resolving multiple responses,[6] the accession number of a particular responding document is generated. Then the binary serial numbers of the English words contained in this document index are generated one at a time. By means of the Descriptor Selector, each serial number is transmitted to the Dictionary MIRF, where it is translated to the alphabetic code of the English word. This process is repeated for each responding document.

SYSTEM DESIGN

1. Magnetic Implementation of the MIRF Unit

The MIRF units of the experimental model use an interesting modification of the Dimond Ring[7] translator in which the drive and sensing functions are interchanged. Information is stored in unique wiring patterns associated with an array of linear ferrite cores as il-

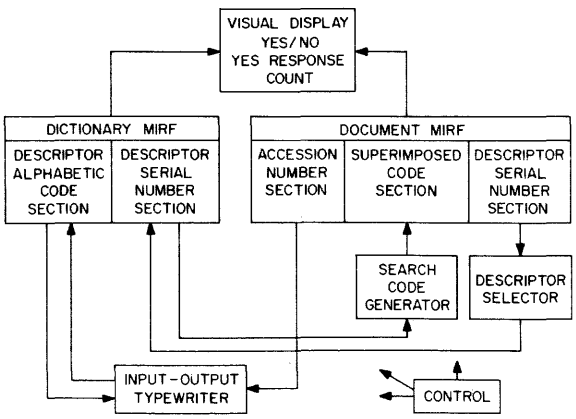


Figure 1. Simplified Block Diagram of MIRF Experimental Model.

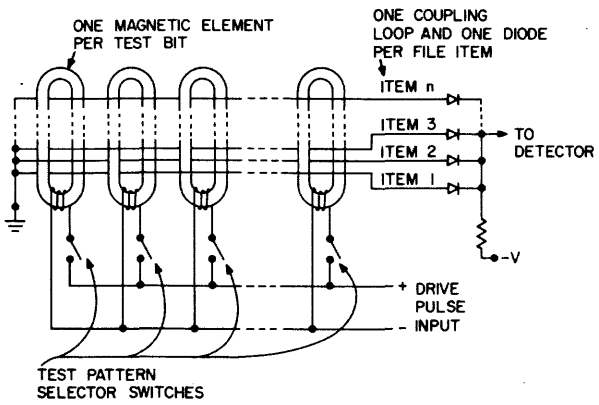


Figure 2. Core-Wiring Arrangement for MIRF Memory.

lustrated by Fig. 2. Each item of stored information (a document index in the Document MIRF or a descriptor in the Dictionary MIRF) is represented by a conductor that passes through or around each associated core in a unique pattern determined by the information it contains. In series with each conductor is a diode. The cathodes of many diodes are connected together to form the input to a detector amplifier. Notice that one core is required for each bit of information, but that each core can be associated with a particular bit of many item conductors.

Each core has an input winding that can be selected by means of a switch. All cores whose selector switch is closed will be energized when a drive pulse is applied. A voltage will be induced in each item conductor that threads an energized core, but no voltage will be induced in conductors that do not thread the core. A test can be made on the information stored in many cores by selecting a particular set of cores and energizing them. In order for an item to match the test information, its conductor must pass outside of every energized core. Then no voltage will be generated in the item wire and the input to the detector amplifier will be held near ground through the item diode. Voltages will be induced in the conductors of items that do not match the test; the polarity of these voltages is chosen to back-bias the associated diodes. If no item matches the test information, a voltage will be induced in every item conductor and every diode will be back-biased. The input to the detector will then assume a significantly negative voltage. Thus, the presence or absence of desired stored information can be determined by applying the drive currents to a particular set of cores. This is a function of an associative or content-addressed memory: to indicate the presence or absence of certain information based on the detailed contents of a search question without regard to the actual location (or address) of that information.

Now consider in more detail how a bit of information of a search question is compared with information in a MIRF unit. Figure 3 illustrates how a test is made to determine whether or not the test bit is logically "included" in the stored information. This circuit is typical of those used in the superimposed section of the Document MIRF. One core is used to store the kth bit of many items. The kth bit of the search question is stored in a flip-flop whose *one* side is connected by way of an AND gate to a drive amplifier, which in turn is connected to the primary winding of the kth core. The conductor of an item whose kth bit is equal to *one* (Conductor 1) passes outside the kth core. On the other hand, the conductor of an item whose kth bit is equal to *zero* (Conductor 2) threads the core. If the flip-flop stores a *one*, the primary winding of the core will be energized when the timing pulse is applied to the AND gate. A voltage will be induced in Conductor 2 (indicating a mismatch) but none will be induced in Conductor 1 (indicating a match). If the flip-flop stores a *zero*, the primary winding will not be energized because the timing pulse will be blocked at the AND gate. No voltage will be induced in either conductor, and a match will be indicated on both lines. Therefore, it can be seen that a stored *one* bit includes both a test *one* and a test *zero*, while a stored *zero* bit includes only a test *zero*.

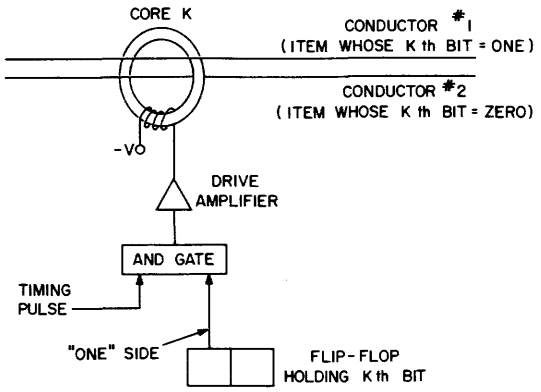


Figure 3. Circuit for Testing Inclusion.

The circuit for testing for identity between the test bit and the information stored in the MIRF is shown in Fig. 4. This circuit is typical of those used in the alphabetic descriptor portion of the Dictionary MIRF. The jth bit of many items is stored in a pair of cores j_A and j_B. The jth bit of the test question is stored in a flip-flop. In this case, both the *one* and *zero* sides of the flip-flop are connected to AND gates

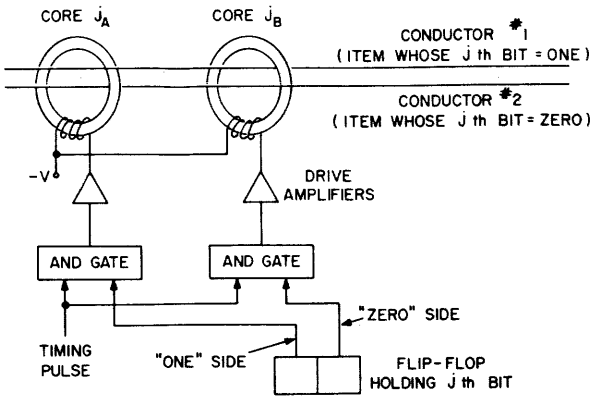


Figure 4. Circuit for Testing Identity.

whose outputs control drive amplifiers that are connected to the primary windings of the cores j_A and j_B. The conductor of an item whose *j*th bit is *one* (Conductor 1) bypasses core j_A while the conductor of an item whose *j*th bit is a *zero* threads core j_A. The threading of core j_B by the two conductors is the reverse of the wiring of core j_A. If the flip-flop stores a *one*, the primary winding of core j_A will be energized when the timing pulse occurs. No voltage will be induced in Conductor 1 (a match indication) but a voltage will be induced in Conductor 2 (a mismatch indication). If the flip-flop stores a *zero*, the primary winding of core J_B will be energized. In this case, a voltage will be induced in Conductor 1 but not in Conductor 2. Thus it can be seen that the bit stored in the MIRF must match the test bit identically for a match indication to be obtained.

2. *Basic Operations Using the MIRF Units*

Two types of operations involving the MIRF units are basic to the operation of this experimental model. One operation tests to see if certain information is contained in the MIRF. The other uses information that is contained in the MIRF to generate a number in a flip-flop register external to the MIRF unit. Examples of these basic operations are given in the following paragraphs.

a. *Testing of Information Contained in the MIRF Unit*

Dictionary MIRF—During the input of the English words to form a search question, the Dictionary MIRF is tested to see if the input word is contained in the vocabulary (that is, if it is a valid descriptor). This is done by gating the alphabetic descriptor register to the drive amplifiers associated with the alphabetic portion of the MIRF (50 bits long, two cores per bit). As a result, 50 drive amplifiers are energized and 50 primary windings in the MIRF carry current. If one of the stored words has a bit pattern in the alphabetic portion that matches identically the energized set of primaries, the match detector will indicate a match condition. If not, the match detector will indicate a mismatch condition. The output of the match detector is used to determine the next step in the logical sequence. It is important to note that the test is applied to the entire Dictionary MIRF simultaneously and that a match or mismatch signal for the entire MIRF is obtained in about 5 microseconds.

Document MIRF—After all words of the search question have been typed, the *superposition* of their search codes is held in the search code accumulator. At the beginning of the actual search operation, the flip-flops of the search code accumulator are gated to their associated drive amplifiers. A particular set of drive amplifiers is energized and current flows in a corresponding set of primary windings in the 80 bit superimposed code field of the Document MIRF. If the detailed bit pattern represented by the energized primaries is included in any of the superimposed fields of the stored document indexes, a match condition is indicated by the match detector. If not, a mismatch indication is given. The test is made on the entire contents of the document MIRF simultaneously and a YES/NO response is obtained in about 5 microseconds.

It should be pointed out that the criterion for a match is inclusion, not identity. A document index includes the search question if the following conditions of the superimposed search code portion of the index are satisfied. First, for every bit of the index search field that is a *one*, the corresponding bit of the search question is either a *zero* or *one*. Second, for every bit of the index search field that is a *zero* the corresponding bit of the search question is a *zero* (in other words a binary *one* includes both

a *one* and a *zero*, but a binary *zero* does not include a binary *one*).

b. *Generating Numbers by the MIRF Process*—The generation of the serial number of an input descriptor illustrates this operation. Assume that an English word has been typed in and that the test for valid descriptor is true. Because a match is obtained when the alphabetic descriptor register is gated to the Dictionary MIRF, one item wire in the MIRF is effectively isolated: namely, the wire that is uniquely related to the input descriptor. The detailed wiring pattern of this wire in a group of cores outside the alphabetic code field contains the binary serial number of the input descriptor. By gating the alphabetic descriptor register to the MIRF and at the same time causing current to flow in the primary winding of a core that is in the serial number portion of the MIRF, the binary value associated with that core for the selected line can be determined. The presence of current in the additional winding tests for a binary *one* in that position. If the match detector indicates a match, the value is indeed *one*. However, if a mismatch is obtained, the value must be *zero*.

The sequence for generating the serial number is as follows: First the flip-flop register that will eventually hold the serial number is cleared to all *ones*. Then the alphabetic descriptor register is gated to its drive amplifiers and a drive amplifier associated with the parity bit of the serial number is energized. The output of the match detector is observed. If a match condition is observed, it is known that the parity bit is actually a *one* and the parity bit flip-flop in the serial number register is not changed. If a mismatch is observed, it is known that the parity bit is *zero* and the parity bit flip-flop in the serial number register is not to *zero*. The next step is to energize the drivers associated with the alphabetic descriptor register and a driver associated with the least significant bit of the serial number. Again the output of the match detector is observed and the flip-flop assigned to the least significant bit is either allowed to stay at *one* or is changed to a *zero*. This procedure continues for thirteen steps. At the end of this time, the 12-bit serial number and its parity bit will have been generated and stored in the serial number register.

CIRCUIT DESIGN

Three principal types of transistor circuits are used in the experimental model: transistors are used as switches to drive the primary windings of the MIRF cores; discriminator-amplifier circuits are used to accept the voltage generated on the secondary windings of the MIRF cores (this is the match detector circuit); and transistor logic circuits are used for the over-all control of the MIRF operations. All three types were designed at SRI.

1. MIRF Driver

The drive currents that are required by the ferrite cores in the Document and Dictionary MIRFs are furnished by circuits such as the one shown schematically in Fig. 5. Four MIRF driver circuits are mounted on one printed circuit plug-in board, as shown in Fig. 6. Each circuit is capable of supplying the required 2 amperes at low impedance. The power transistor that delivers the drive current (Type 2N1905) is driven by a push-pull emitter follower that provides 60 milliamperes of base drive current into 2N1905. The output power transistor has rise-and-fall time capabilities of less than 0.3 microsecond. The actual current in the load is nearly linear because of the inductive nature of the load and builds up to the 2 ampere amplitude at the end of approximately 10 microseconds. The overshoot voltage induced when the transistor is turned off is clamped by a silicon diode to —36 volts. The clamp prevents excessive voltage spikes from appearing across the output transistor while still allowing the load inductance to recover within 10 microseconds.

Two protective features of the MIRF driver circuit should be noted. One is a fuse, which is inserted in series with the load to protect against excessive load currents. Before the winding of the magnetic circuits internal to the MIRF assembly can be damaged by too much current from, say, an accidental short circuit, the fuse wire will open up. The second protective circuit includes a square-loop memory core that is threaded by the lead going to the transistor load. This core is normally biased off, but if the drive current exceeds a safe value the square-loop core will switch and induce a voltage in a sense lead. The voltage in

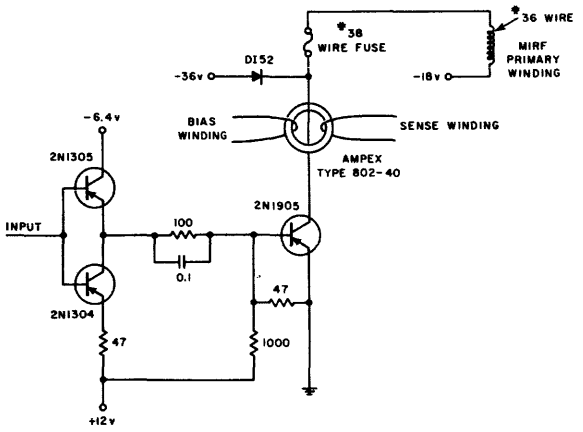


Figure 5. Schematic of MIRF Driver.

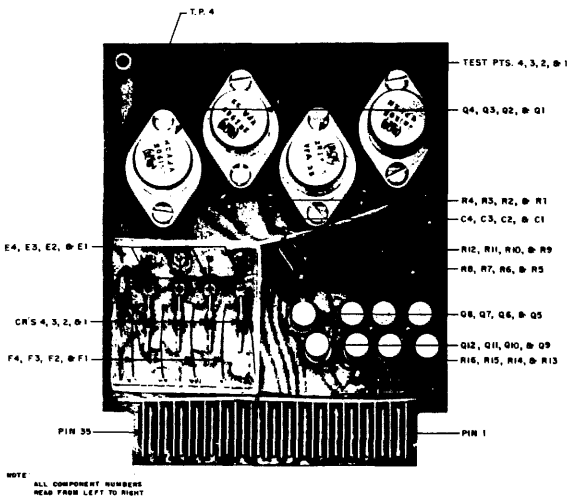


Figure 6. Component Assembly of MIRF Driver Board.

the sense lead is amplified and used to turn off the system clock. The purpose of this circuit is to protect the 2N1905 transistor against excessive heat dissipation from currents that are excessive but not large enough to burn out the fuse wire.

2. *MIRF Discriminating Amplifier*

The electrical output of the MIRF magnetic modules is generated by a very large diode gate including almost 300 diodes. Under the worst conditions a match signal from this array can reach a level as high as 0.4 volt. On the other hand, a mismatch signal from the same array may only generate a potential of 0.6 volt. It is necessary for the MIRF discriminating amplifier to differentiate between these two signals and generate a standard logic level output of —6 volts for a mismatch and 0 volts for a match. The circuit for the amplifier is shown in Fig. 7. In order to distinguish between very closely spaced match and mismatch signals, two thresholds are employed in the amplifier. The first threshold is provided by a 1N3605 silicon diode at the input to the amplifier. This diode does not pass signals unless they exceed approximately 0.5 volt. After passing the first threshold, the signal is amplified in a feedback amplifier with a gain of about 50. If the amplified signal then exceeds the second threshold of 3 volts, a mismatch signal is delivered at the output of the amplifier.

3. *Logic Circuits*

In the flip-flop register and over-all control circuits, resistor-transistor logic is used. Highly reliable circuits that operate in the 100-kc frequency range have been developed. The basic gate circuit is shown in Fig. 8. This circuit in typical use performs a simple majority operation. If one or more of its three inputs are at a negative potential, the output is held at ground potential. Since ground is defined as the *one* state in this system, and a —6 volt potential is defined as a *zero* state, the basic gate performs the "not and" or NAND operation.

All the passive components shown in Fig. 8, plus one resistor and two capacitors, are contained in one physical element supplied by Centralab, Inc. These components are screened on a passive substrate to a tolerance of 3% for the resistors (5% design tolerance) and

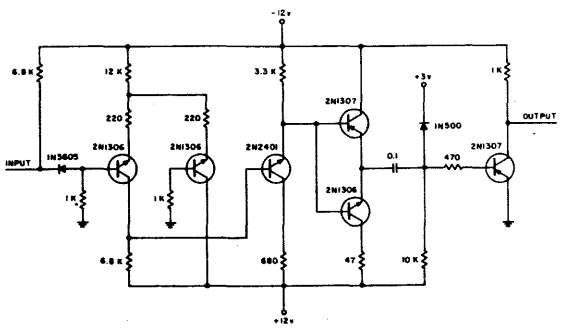


Figure 7. Schematic of Discriminating Amplifier.

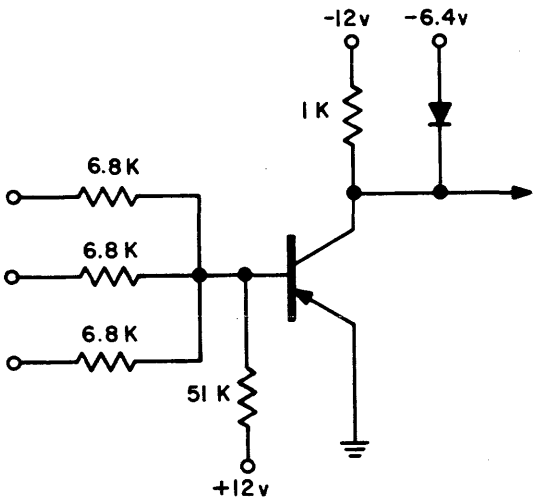


Figure 8. RTL Circuit Designed for the MIRF System.

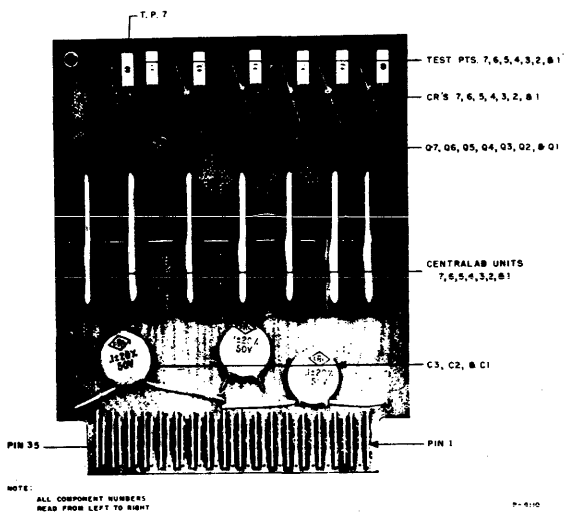


Figure 9. Component Assembly of Gate-Logic Board.

10% for the capacitors. The substrates are encapsulated with a Durez coating, and are ready for mounting to a printed circuit card via their projecting leads.

The gate circuit is a basic part of every logic circuit employed in the machine. By itself it performs the combinatorial function of logical conditions. Two gate circuits properly interconnected form a bistable, or flip-flop, circuit. Two gate circuits interconnected in a slightly different way form a monostable, or one-shot, circuit. The gate circuit is also used as a preamplifier for an emitter-follower circuit. The basic logic circuits, e.g., gates, one shots, flip-flops, etc., are mounted on plug-in logic boards. A typical logic board, with seven gate circuits mounted on a printed circuit board, is shown in Fig. 9.

MAGNETIC DESIGN

1. General Considerations

The magnetic design of a MIRF unit is centered in the individual magnetic core, which acts as a transformer with a multiturn primary winding and many single-turn secondary windings. When current flows in the primary winding, the magnetic core must be capable of producing a flux change of sufficient time duration and amplitude to generate the desired signal in secondary windings. The amplitude of the induced voltage is determined primarily by the characteristics of the diode associated with the secondary winding. The duration of the induced voltage is determined primarily by noise on the secondary winding and the consequent delay required before sampling of the output can be accomplished.

The cross-sectional area of the magnetic core is proportional to the product of the amplitude and duration of the voltage induced in the secondary windings (this is usually referred to as the *volt-second area* of the induced voltage pulse). This was kept reasonably small by using a high-quality germanium diode (the 1N500) which requires a back-biasing voltage of only 0.6 volt in order to perform properly in the diode circuit associated with the input to the discriminating amplifier. The circumferential length of the magnetic core is determined primarily by the number of secondary windings associated with the core and the mechanical design of the supports for these windings. In the MIRF units of the experimental equipment, the core has the capacity for 2,000 secondary windings. The core's mean circumferential length is 7 inches; its cross section is a square, ¼ inch on a side.

Two other considerations influenced the selection of the magnetic cores used in the MIRF units. One is the requirement that the core be made in two pieces so that the array of cores

can be separated into two portions to facilitate initial wiring and changes in wiring. The other is the necessity of using commercially available parts. The number of cores needed in this experimental equipment is too small to justify the design and production of a core of special size or shape.

2. Details of the Dictionary and Document MIRF Units

The individual cores used are the same for both the Dictionary and Document MIRF. Each core is composed of two U-shaped ferrite structures (Allen Bradley part no. UC 892–141C), which have been specially modified at the factory to permit a maximum of 0.0005 inch air gap in each leg when two such structures are joined together to produce a MIRF core. To drive each core, a twenty-turn primary winding is provided. This consists of two ten-turn windings distributed in such a manner as to minimize the leakage flux and the resulting noise signal (see Fig. 10). The primary winding drives the core from an 18-volt voltage source through a transistor switch driver. The output voltage induced upon each secondary winding is an essentially rectangular voltage pulse having a droop of 0.1 volt in 10 microseconds, from 0.8 volt at the leading edge to 0.7 volt just prior to the trailing edge. The maximum primary current, 0.7 ampere, occurs at 10 microseconds after the beginning of the pulse. To accommodate the expanded capacity of the MIRF document file (5,000 documents) three primary windings will be driven in parallel, so that a maximum driver current of 2.1 amperes is required.

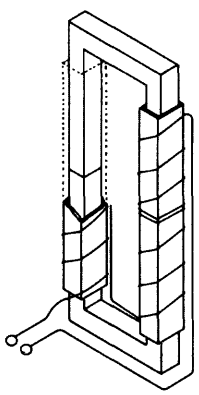


Figure 10. Details of Primary Windings.

The performance requirement of the magnetic circuits is that consistent and easily separable match and mismatch signals be generated at the diode end of the item wires (see Fig. 2) when a set of primary windings is driven. The design objective was that a maximum match signal of 0.1 volt and a minimum mismatch signal of 0.6 volt should be realized within 1.5 microseconds after the application of the primary drive pulses, and that pulsing of the MIRF cores be repeated for many cycles at a 50-kc clock rate. To achieve these goals, noise due to ringing and leakage flux had to be minimized.

A MIRF unit contains many cores (the Document MIRF has 234 and the Dictionary MIRF has 140), each with a separate primary winding; further, each core is associated with more than a thousand single-turn secondary windings. The secondary windings pass through or around all cores in the unit and so form a long rope. The capacitance between wires in the rope, the inductance of these wires, and the inductance of the primary windings are intercoupled in a very complex manner. In the development of the MIRF units, substantial noise on the secondary (item) windings was experienced due to ringing currents in the primary windings. This noise was reduced to a negligible level by inserting a Type DI52 diode in series with each primary winding and shunting each primary by a 1000 ohm resistor. A low-amplitude noise signal of about 5 Mc, due to inductance and inter-item capacitance of the secondary windings, was also observed. Such noise could be reduced to a very low level by filtering at the input to the discriminating amplifier, but in the experimental system this was not necessary.

Noise due to leakage flux must be kept small in order to hold the maximum match signal at 0.1 volt. A secondary wire that represents a match item must pass outside all energized cores. Since in the worst case, 57 cores may be energized, the maximum permitted noise due to leakage flux at each core is less than 2 millivolts (this corresponds to a leakage flux of ¼ of one per cent at each core). In the experi-

mental model two methods are used to reduce leakage flux. One is distributing the primary winding on the cores to compensate for the magnetic potential drop by a corresponding rise in magnetic potential at the points where the drop occurs. As Fig. 10 shows, the winding has a linear spacing except at the points where the air gaps occur; there two turns are closely spaced. The second method uses cancellation of induced voltages to reduce the effect of leakage flux. The common end of many item wires, instead of being connected to ground, as shown in the simplified diagram of Fig. 2, is actually connected to a wire that lies in the item wire rope and passes *outside* of all cores. The voltage induced in the "cancellation lead" at any core by leakage flux is approximately equal to that induced in item wires and is opposite in polarity (relative to the input terminals of the discriminating amplifier).

MECHANICAL DESIGN

1. *The MIRF Module*

Implementing the wiring-patterns-on-cores method of storage illustrated by Fig. 2 presented a challenging mechanical design problem. It was necessary that the physical structure containing the magnetic cores and the associated wiring be made in two parts that could be easily separated. It was desirable to fabricate submodules of wiring patterns, so that the permanently stored information could be changed mechanically in relatively small blocks.

Separate MIRF modules are used to store the information concerning document indexes and dictionary words. In each, the cores are arranged in a rectangular pattern and are supported by long bobbins. These bobbins are firmly attached to a base structure and carry the primary windings for the cores. A MIRF module is a complete assembly of magnetic cores, primary windings for the cores, and submodules of secondary windings with their associated diodes. The construction of a module is illustrated by the exploded view of Fig. 11. The principal parts of the assembly are the base, or coil bobbin, assembly and the item wiring trays.

The coil bobbin assembly consists of a field of paper bobbins (two per magnetic core) that are cemented to a $\frac{1}{8}$-inch-thick phenolic board. Each bobbin carries a ten-turn winding. The windings on pairs of bobbins are connected in series to form the primary winding for one of the magnetic cores. An item tray is a $\frac{1}{16}$-inch thick phenolic board with a field of shallow bobbins that matches the field of coil bobbins. The bobbins on the item tray are slightly larger than the coil bobbins, permitting item trays to be stacked up on the coil bobbin assembly. One item tray can accommodate 286 item wires. The diodes that are connected in series with the secondary windings and form the input circuit to the discriminating amplifier are mounted on the edge of the item tray. A MIRF module is assembled by sliding up to seven item trays into position on the coil bobbin assembly. One set of U cores is then inserted into the set of coil bobbins and held in place by a plate with a silicone-rubber pad. The other set of U cores is then dropped into position on the opposite side of the bobbin coils. Finally, the top plate (also with a spongy pad) is dropped into position to hold the entire assembly intact. The two sets of U cores are held together under slight pressure from the silicone pads.

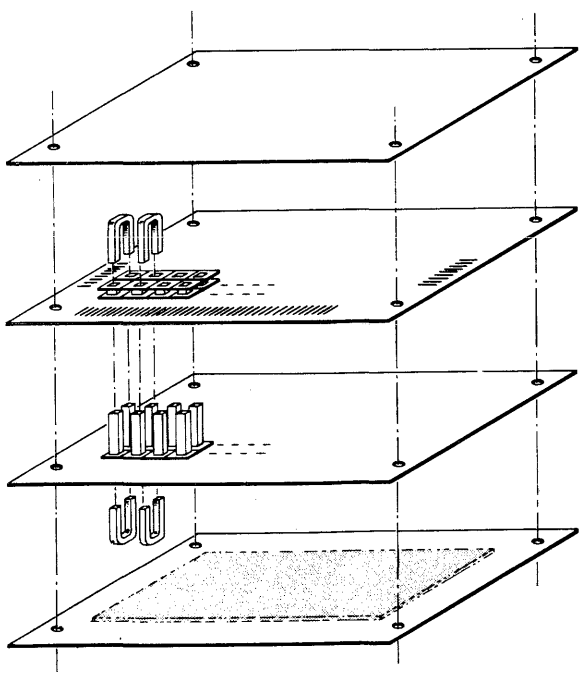


Figure 11. Exploded View of MIRF Module.

A complete item tray is shown in Fig. 12. The item wires start in the upper left corner of the trays, where they are connected to a common bus bar. They pass from left to right in the first row of cores, then back and forth until they emerge in the lower left center part of the tray. The wires then run to assemblies of diodes, where each wire is connected to its own individual diode. The output side of the diodes (the cathodes) are connected together and wired to a small connector, which is seen in the lower left hand portion of the tray. Even though each tray contains detailed wiring for 286 items, only two wires run from the tray to the external discriminating amplifier. Figure 12 also shows a pair of primary coil bobbins with the two U cores inserted. A closeup of a MIRF module with the top plate removed is shown in Fig. 13. The tops of one set of U cores can be seen as well as four item trays. The connectors for the output of the item trays can be seen in the lower center part of the photograph. The discriminating amplifier circuits (one for each of the seven item trays that can be included in a module) are located on the circuit board that is mounted in front of the magnetic module.

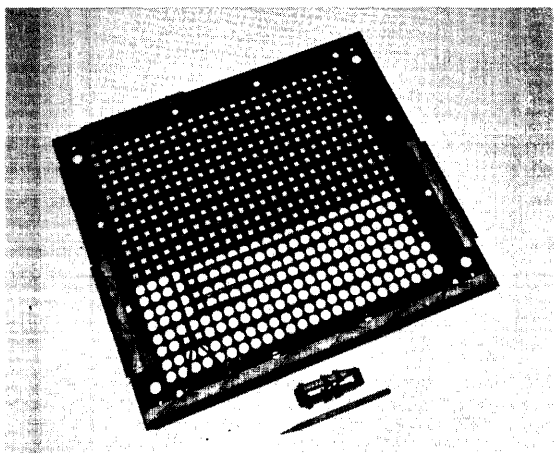


Figure 12. MIRF Item Tray.

2. *Wiring of the Item Trays*

The item trays in the Document and Dictionary MIRF units store more than one-third of a million bits of information. To ensure the greatest possible accuracy of the wired-in information, two steps were taken. First, the raw data for the documents were computer-processed to give a set of punched cards that contain the detailed wiring information. Second, a wiring scheme was devised, which presented the detailed wiring information to a wireman in a very simple form, and which included a means of checking the accuracy of the wiring as the wiring was actually done. In this scheme, the path that a wire was to take was delineated by a set of lights in an array of incandescent lamps.

An over-all view of the item-tray wiring equipment (wiring aid) is shown in Fig. 14. The empty wiring tray is placed on the wiring jig in front of the operator. A card is then

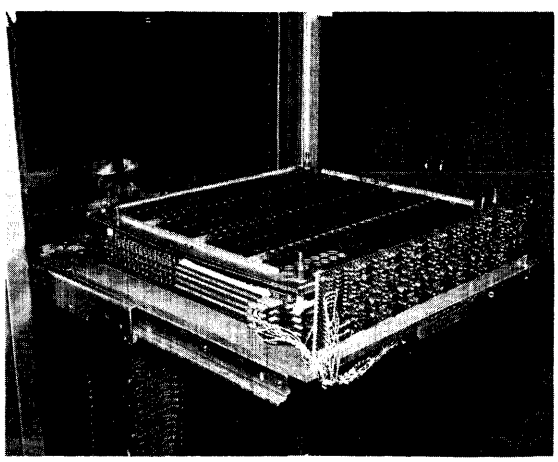


Figure 13. Close-up of Document MIRF Module (Top Plate Removed).

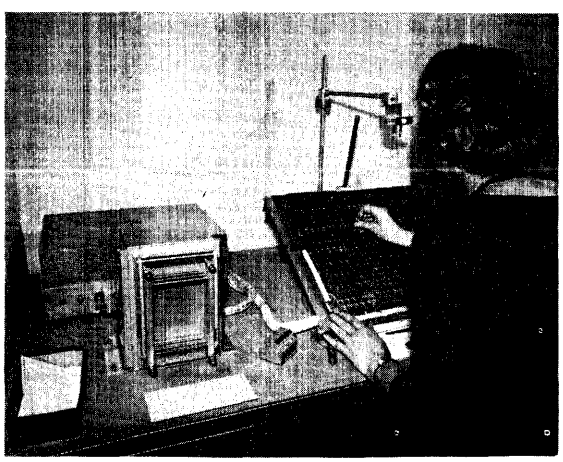


Figure 14. Over-all View of Item Tray Wiring Equipment.

placed in the punched-card reader and a pattern of lights is set up in the wiring jig. Number 36 Nyleze wire is taken from a spool through a tensioning device to the top of a special wiring tool (shown in the hand of the operator). The wire from the bottom of the wiring tool is first soldered to the common bus shown in the upper left part of the wiring tray. The tool is then moved along the path specified by the pattern of lights, leaving the wire wound in the desired pattern around the item tray bobbins. Correct wiring at a bobbin is indicated by a light turned on to yellow brilliance. If a light is off, or is on at white brilliance after the wiring tool passes a bobbin position, a wiring error is indicated.

3. *Alternative Method of Fabricating Item Trays*

Alternative methods of preparing wired-in information that may be more easily automated than stringing of small wire have been investigated. One alternative is illustrated by Fig. 15, which shows an item conductor in the form of a metallic path etched on a thin, copper-coated Mylar sheet (half-ounce copper on 2-mil Mylar). It will be noted that the item conductor is connected to a bus at the top of the sheet and to another bus at the bottom. These copper areas are used for connecting the item conductor to the common bus at one and to a diode at the other. This sheet contains one item, but two item conductors could easily be placed on one sheet, one being associated with one leg of the magnetic core and the other with the other leg. The experimental model contains a submodule of 75 items on Mylar sheets.

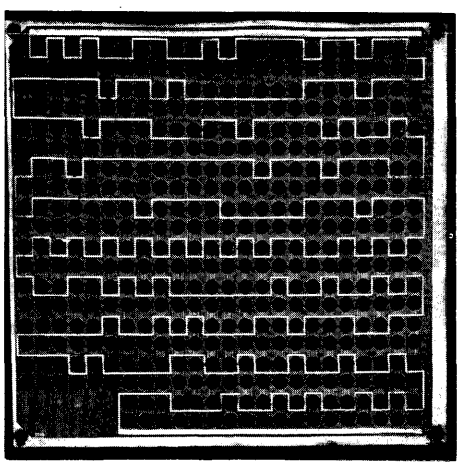


Figure 15. MIRF Item Conductor Formed by Metallic Path on Mylar Sheet.

DELIVERED EXPERIMENTAL EQUIPMENT

The experimental Multiple Instantaneous Response File System is an all-solid state equipment. Transistor drive circuits capable of supplying two amperes of current to magnetic circuits, special discriminating amplifiers capable of operating reliably with a poor signal-to-noise ratio input signal, and transistor logic circuits were designed for high reliability, low cost, and moderate speed. About 300 current drive transistors, 2500 logic transistors, 2500 printed gate circuits (a group of 6 resistors, 2 capacitors and their interconnecting wiring on a passive substrate) and 5,000 diodes are used in the system. Except for sequences involving the input-output typewriter, the system operates synchronously under the control of clock pulses derived from a 50-kc transistor multivibrator.

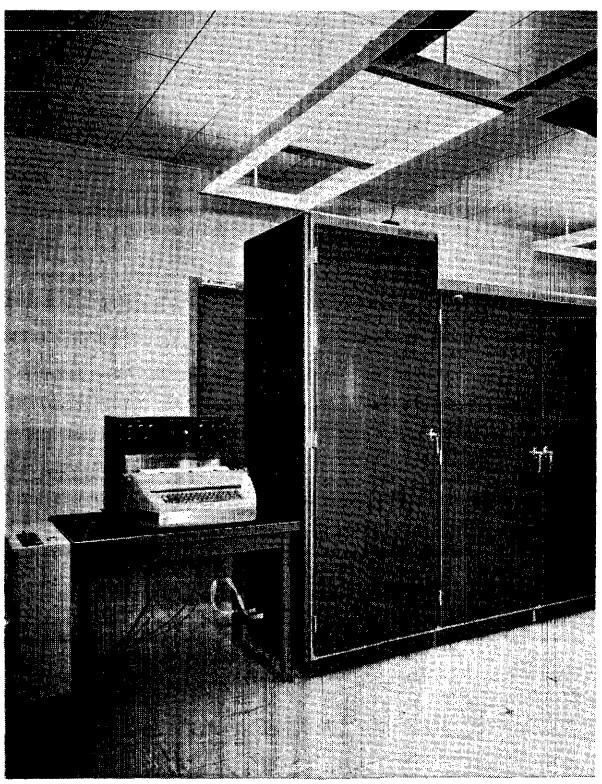


Figure 16. Front View of Experimental MIRF Equipment.

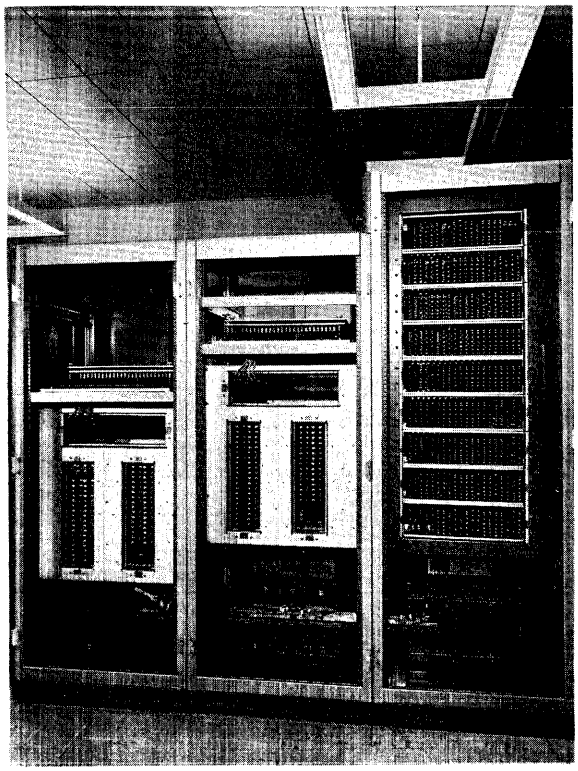


Figure 17. Rear View of Experimental MIRF Equipment (Doors Removed).

The experimental equipment shown in Figs. 16 through 18 was delivered to Rome Air Development Center in July, 1963. A front view of the equipment is shown in Fig. 16. The main equipment cabinet, the input-output typewriter, and the display and control unit can be seen. Figure 17 shows a rear view of the equipment cabinet with the doors removed. The right hand portion of the cabinet contains logic circuits for control of the system, arranged in modules of plug-in transistor logic boards. The Dictionary MIRF unit is contained in the center portion of the cabinet. Directly beneath the MIRF unit are two modules of drive circuits which provide current to the MIRF. In the left hand portion of the cabinet are the Document MIRF and the transistor circuits for providing drive currents to it. It will be observed that space has been allowed for one additional MIRF unit in the center section and for two additional MIRF units in the left hand section. This is to provide for the expansion of the Dictionary MIRF to 3,000 words and expansion of the Document MIRF to 5,000 document indexes. A

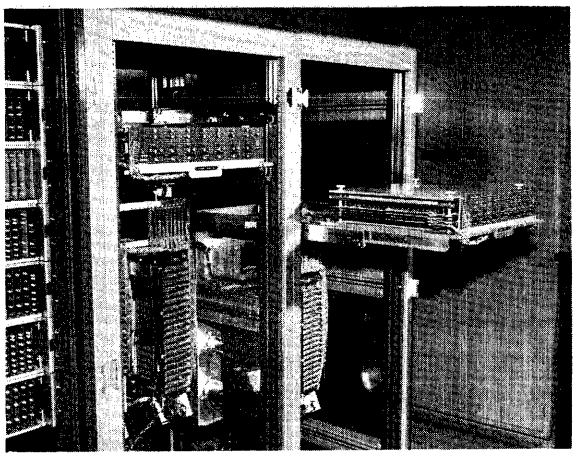


Figure 18. Front View of Equipment with Document MIRF Module in Extended Position.

front view of the cabinets that house the MIRF units and their drivers is shown in Fig. 18. Here the Document MIRF unit has been pulled out to show it in its extended position. Below the MIRF units the wiring side of the transistor drive modules can be seen.

The format of the typewritten record of a search in the experimental model is shown in Fig. 19. The first two lines, "Stanford Research Institute Project 4110," etc., are a manually typed heading for the subsequent search. The heading was typed while the typewriter was effectively disconnected from the rest of the equipment. The search question consists of three words: "coding," "computers," "digital." This line was also typed manually. The rest of the printout is the machine's response to the search question. Seven documents responded. For each one, a four-digit accession number and the English words that describe the document are printed on a single line. The asterisk prefix on some words have been copied from the ASTIA abstract. It will be observed that the three search words appear in every respond-

STANFORD RESEARCH INSTITUTE PROJECT 4110

MULTIPLE INSTANTANEOUS RESPONSE FILE

CODING, COMPUTERS, DIGITAL.
0156 *CODING, DIGITAL COMPUTERS, DATA PROCESSING SYSTEMS, LANGUAGE,
0201 RADAR PULSES, RADAR SIGNALS, *CODING, DIGITAL COMPUTERS,
0420 DESIGN, DIGITAL COMPUTERS, *LANGUAGE, CODING, ANALYSIS,
0540 DIGITAL COMPUTERS, ERRORS, LANGUAGE, CODING, MATRIX ALGEBRA,
0727 *LANGUAGE, *CODING, *HANDBOOKS, DATA PROCESSING SYSTEMS, DIGITAL COMPUTERS,
0732 DIGITAL COMPUTERS, CODING, TELETYPE SYSTEMS, DISPLAY SYSTEMS, MAPS,
0824 DATA PROCESSING SYSTEMS, DIGITAL COMPUTERS, OPERATIONS RESEARCH, CODING,

Figure 19. Format of Typewritten Record of a Search.

ing set of indexes. It should be especially noted that the search words appear in different positions and different order in the different responding documents. This independence of order of the search words and the position of the corresponding descriptors in the document indexes is an important result of the superimposed coding of the search field.

CONCLUSIONS

From experience with the Experimental MIRF it is concluded that interrogation of the magnetic storage units and the over-all control of the system can be accomplished with reliable circuits of modest complexity. Storage of the document index information in wiring associated with arrays of cores that are physically separable appears feasible; arrays of cores can be separated, submodules of wired information can be changed, and the core arrays reassembled in a reasonably short time. More work on the mechanical design of the magnetic modules is needed, however, to permit easier and faster changing of the stored information. Based on the performance of the experimental model, which contained a file of more than 1,000 document indexes, it is concluded that with the present design a system building block should contain about 5,000 document indexes. It appears that as many as ten such building blocks could be combined in a system whose over-all control is little more complex than that for a single building block. Therefore it is concluded that files of the order of 50,000 indexes could be built with no major changes in the basic concepts or circuits used in the experimental model.

Easy communication between a human operator and the Experimental MIRF System has been demonstrated. The machine's response to a search question is essentially instantaneous in terms of human reaction time and the information content of the response is sufficient to allow the operator to start the document search with a general question and to use the information received to define a more specific question. In this way it is possible to home-in quickly on the documents of special interest. Several automatic features of the equipment have proved to be useful. One of these is the capability of accepting a synonym in the search question and automatically translating it into the synonymous descriptor contained in the machine's vocabulary. Another feature is the capability of automatically modifying the search question inserted by the human operator and initiating a new search. For example, if any of the input words have attached to them a "see-also" reference, that see-also reference will be substituted for the original word to form a new search question.

ACKNOWLEDGEMENTS

The development of the Experimental Multiple Instantaneous Response File was sponsored by the U. S. Air Force, Rome Air Development Center, under Contract No. AF 30 (602)–2772. The authors wish to acknowledge the contributions of their colleagues at Stanford Research Institute, especially C. B. Clark, D. C. Condon and V. Sanford.

REFERENCES

1. A. E. SLADE and C. R. SMALLMAN, "Thin Film Cryotron Catalog Memory," *Proc. of the Symposium on Superconductive Techniques for Computing Systems,* Washington, D. C., May 1960, published in *Solid State Electronics,* vol. 1, pp. 357–362 (September 1960).

2. J. R. KISEDA, H. E. PETERSEN, W. C. SEELBACH, and M. TEIG, "A Magnetic Associative Memory," *IBM J. of Res. and Dev.,* vol. 5, pp. 106–121 (April 1961).

3. J. GOLDBERG and M. W. GREEN, "Large Files for Information Retrieval Based on Simultaneous Interrogation of All Items," *Large Capacity Memory Techniques for Computing Systems,* M. C. Yovits, Ed., pp. 63–77, MacMillan Co., New York, 1962.

4. M. H. LEWIN, H. R. BEELITZ, and J. A. RAJCHMAN, "Fixed Associative Memory Using Evaporated Organic Diode Arrays," *AFIPS Conference Proceedings,* vol. 24, pp. 101–106 (November 1963).

5. E. L. YOUNKER, D. C. CONDON, C. H. HECKLER, JR., D. P. MASHER, and J. M. YARBOROUGH, "Development of a Multiple Instantaneous Response File—the AN/GSQ–81 Document Data Indexing Set," to be published as a Rome Air Development Center Technical Documentary Report.

6. E. H. FREI and J. GOLDBERG, "A Method of Resolving Multiple Responses in a Parallel Search File," *IRE Trans.*, EC–10, pp. 718–722 (December 1961).

7. T. L. DIMOND, "No. 5 Crossbar AMA Translator," *Bell Labs Record*, vol. 29, pp. 62–68 (February 1951).

RESEARCH IN AUTOMATIC GENERATION OF CLASSIFICATION SYSTEMS

Harold Borko, Ph.D.
System Development Corporation
Santa Monica, California

INTRODUCTION

This paper is concerned with the organization of information, in the form of documents, for efficient storage and retrieval. By documents we mean books, technical reports, articles, memoranda, letters, photographs, data facts, etc.—all forms of memory file organization ranging from documents in a library to data in a real-time command-and-control system. Therefore, the implications of this work are applicable to a field broader than the concerns of the ordinary library.

In actual practice, most of the information retrieval research has been concerned with document files because the most highly organized collection of documents in existence today is the library, and in doing research on methods of organizing information, one must compare the adequacy of proposed new techniques with existing library methods. Procedures which will improve information storage and retrieval in a library will probably be sufficiently powerful to help improve other methods of file organization.

PURPOSES OF DOCUMENT CLASSIFICATION

The reason for maintaining a collection of documents is to have an available store of information and to be able to retrieve desired information rapidly and with confidence. The value of classification is that it increases efficiency in locating this desired information. If we tried to locate a book on a particular subject in a library that did not use any system of classification, we would have to spend a long time reading the titles and authors of several thousand books before we could find the one for which we were looking. If we knew the author, and the books were arranged alphabetically by author, we could locate the book quickly. On the other hand, if we didn't know the author, but knew the subject content of the book, we would want the books arranged by subject category in order to search the file efficiently. Finally, if all we knew was that the book we sought was a big black one which we could recognize, we would like the files arranged by color. The point being made is that there are various ways of organizing a file, and whether or not a particular method of file organization is efficient depends upon the search strategy. Furthermore, no one method of file organization would be equally efficient for all search questions. This is an important, if obvious, point and one which is often overlooked.

The central theoretical problem of classification as a method of organizing documents is that only one principle at a time can be utilized for gathering items together. This principle can be alphabetic arrangement by author, color coding based on the binding of the book, subject classification, or and other scheme—as long as only one principle is used at a time.

Since a document collection is a store of information, it is usually desirable to organize this store according to subject matter. By establishing clearly demarcated groups, or classes, of documents on related topics, the number of documents to be scanned can be reduced to reasonable proportions. This, in essence, is the purpose of classification. A classification system

is a scheme for organizing a mass of material into groups so that related objects are brought together in a systematic fashion. Objects in one group are selected so as to be more like each other than objects in any other group. However, before this aim can be realized, two questions must be answered:

1) How many classes shall be established?
2) What shall be the measure of similarity and, hence, what is the principle to be used in determining class membership?

If one is interested in automatic procedures, one has to answer a third question, namely:

3) What principles of classification are most amenable for use in an automated document classification system?

All three of these problem areas are being studied, and some results are already available.

DEVISING A CLASSIFICATION SCHEDULE

The classification of knowledge is not a new problem. Even in ancient times, man sought to organize information of the world around him into categories for efficient retrieval. What is new, perhaps, is the application of mathematical techniques to the classification problem. The older forms of classification, from ancient times through the Dewey Decimal System, were attempts to impose logical subdivisions on the whole field of knowledge. The surprising thing is not that these systems were imperfect, but rather that they succeeded as well as they did. Melvil Dewey first proposed his Dewey Decimal System in 1876, and it is still in extensive use. Now, as a result of new inventions and accelerated research, the traditional boundaries between the sciences are breaking down. It is time to reexamine the concept of classification, to go back to basic principles and to study the various methods of deriving a classification system.

Factor Analysis—Borko

In 1958, Tanimoto[12] published a theoretical paper on the applications of mathematics to the problems of classification and prediction. Specifically, he pointed out how the problems of classification can be formulated in terms of sets of attributes and manipulated as matrix functions. An actual application of matrix mathematics to the analysis of a collection of documents was made by Borko[2] in 1961. The aim of this study was to determine whether it was possible to derive a reasonable classification schedule for a collection of documents by factor analysis,[6] a mathematical technique which enables one to isolate the underlying variables in a domain of events. This method has been used by psychologists to determine the underlying variables of intelligence, personality, creativity, ability, etc.

In Borko's classification study, factor analysis was used to discover the relationship of key content words as they are used in psychological literature. Approximately 600 psychological abstracts were selected for study. These were key punched in their entirety, and by means of a computer program called FEAT[9] (Frequency of Every Allowable Term), a frequency count was made of all words, and 90 tag terms were selected for further analysis. These data were arranged in the form of a matrix consisting of 90 terms and 618 documents. A portion of this matrix is reproduced in Table 1. The number in each cell represents the number of times a given word occurred in a particular document. Table 1 shows that the term "child (children)" did not occur in document number 74, occurred twice in document number 307, and three times in document number 374. The term "level (s)" occurred once in document-numbers 74, 626, and 674 and did not occur in the other documents in the example.

Words	Case(s)	Child(ren)	Factor(s)	Level(s)	Psychology(ical)	School(s)
Abstract # 74	0	0	1	1	1	0
307	1	2	0	0	0	1
321	0	2	1	0	1	0
575	1	2	0	0	1	0
626	0	1	0	1	0	2
647	1	2	0	0	1	0
653	4	1	0	0	1	0
674	1	3	0	1	0	0

Table 1. A Portion of the Document Term Matrix

Based upon the data in the document-term matrix, one can compute the degree of association among the terms as a function of their occurrence in the same set of documents. A measure of this association is the correlation coefficient. This is a decimal number which varies from +1.000 to —1.000. A +1.000 would mean a perfect correlation, namely, that every time word X occurred, word Y appeared in the same document; a zero correlation would indicate no relationship; and a negative correlation would mean that if the word X occurs in a document, then word Y is not likely to occur.

The formula for computing the correlation coefficient is as follows:

$$r_{xy} = \frac{N\Sigma XY - (\Sigma X)(\Sigma Y)}{\sqrt{[N\Sigma X^2 - (\Sigma X)^2][N\Sigma Y^2 - (\Sigma Y)^2]}}$$

By applying this formula and computing the correlation between each of the 90 words with every other word (a total of approximately 4000 correlations), one creates the term-term correlation matrix (Table 2). This matrix expresses the actual associations which occurred among selected words in a sample of documents.

These statistical procedures, preparatory to the factor analysis, are important in demonstrating a method for translating a conglomeration of words and documents into a set of vectors which can be processed mathematically. Factor analysis, when applied to the correlation matrix, enables one to determine the basic underlying variables which account for the relations among the words as expressed in the vectors. It enables us to mathematically determine which words are related and form a set; these sets, in turn, are interpreted as classification categories for grouping the original sample of documents.

	Ability	Achievement	Activity	Analysis	Anxiety
Ability		.272	—.028	.048	.080
Achievement	.272		—.026	—.041	.119
Activity	—.028	—.026		—.002	—.025
Analysis	.048	—.041	—.002		.030
Anxiety	.080	.119	—.025	.030	

Table 2. A Portion of the Correlation Matrix

In the experiment just described, the original 90-column matrix was reduced to 10 vectors which accounted for 62% of the total, and it is assumed most all of the common, variance. These vectors were then rotated mathematically to achieve a simpler and more meaningful structure of the hyperspace. They were then interpreted by the investigator as ten classification categories into which the original sample of 618 psychological reports could be grouped—and by implication, all psychological literature.

To illustrate how the factors were interpreted, let us examine the words which had significant loadings on the first factor.

Term #	*Word*	*Factor Loading*
33	girls	.74
10	boys	.73
70	school	.30
2	achievement	.20
63	reading	.18

There were only five words with significant loading. It is fairly obvious that the concept underlying these terms deals with the achievement of boys and girls in school; consequently, this factor was interpreted as academic achievement of boys and girls in school; consequently, in a like manner. These included factors named

experimental psychology, social psychology and community organization, school guidance and counseling, clinical psychology and psychotherapy, etc.

Thus, we have arrived at answers to the two questions posed earlier in this paper: How many classes should be established, and what shall be a measure of similarity? The application of factor analysis enables one to determine the number of categories which should be established in order to adequately describe a given sample of documents. Furthermore, it provides a statistical technique, or principle, for measuring the similarity of content based upon the co-occurrence of key content terms.

There are many questions still to be answered before one can decide on the usefulness of this technique for classification. These questions include:

1) Are the categories stable; do they hold from one sample of psychological literature to another?
2) Are the categories valid; can all documents be reasonably classified into these categories?
3) Are the categories useful; do they lend themselves to automated document classification?
4) Is the technique a general one; can it be applied to documents other than psychological reports?

Before reviewing the studies designed to answer these questions, it would be well to first examine some other mathematical techniques for deriving classification schedules.

Clump Theory—Parker-Rhodes and Needham

At the Cambridge Language Research Unit in England, Parker-Rhodes was also interested in classification theory and a mathematical basis for forming classes of documents. Interestingly, he considered the use of factor analysis but rejected it on two grounds, one theoretical and the other practical. From a theoretical point of view, Parker-Rhodes claimed that "the statistical type of technique has its place only after we have discovered whatever classification there may be. For then it is up to the statistician to say how nearly the properties of particular elements of the universe are inferable from a statement of the classes to which each belongs. . . . This is quite a different enterprise from that of finding the classes themselves"[10] (page 4). On the practical side, factor analysis is rejected as being incapable of handling "really large universes."

Having decided to avoid the statistical concept of determining the probability of class membership, Parker-Rhodes restructured the problem in terms of locating clumps "in a Boolean lattice representing all possible subsets of the universe." Within a Boolean lattice there are many ways of defining clumps, and in fact, many different clumps are defined. Without getting involved in details, it can be broadly stated that "members of a clump must be more like each other, and less like non-members, than elements of the universe picked at random" (page 9). Thus we see the relationship between the theory of clumps and the theory of classification. The method used for locating clumps within the lattice remains to be worked out. Initial procedures for clumping are described by Needham.[8] Research aimed at improving and testing these procedures is still going on. However, even now these techniques have been applied to a 346 x 346 matrix which is beyond the capabilities of presently available factor analysis programs.

Latent Class Analysis—Baker

The similarity between document classification and the problems inherent in the analysis of sociological questionnaire data was recognized by Baker. He then proposed an information retrieval system based upon Lazarsfeld's latent class analysis.[1] As Baker points out, "The raw data of documents, the presence or absence of key words, is amenable to latent class analysis without modification of either the analysis or the data. The latent classes and the ordering ratios yielded by the analysis provide the basis for a straightforward means of classification and retrieval of documents" (page 520).

The latent class model assumes that the population—that is, the number of documents in the sample—can be divided into a number of mutually exclusive classes. Usually the number of

classes is determined by the investigator, although it is conceivable that this parameter can be determined mathematically. One starts by selecting the key words which characterize each class of documents. Then latent class analysis is used to compute the probability that a document having a certain pattern of key words belongs to a given class.

Baker gives the following example: Let us assume that we have 1000 documents in our file. We are interested in classifying these documents into two classes—those dealing with computer automated instruction and those not directly related to this topic. We select as the key words in our search request the following:

1. computer.
2. automated.
3. teaching.
4. devices.

Each of the 1000 documents are then analyzed to determine whether they contain one or more of the four terms. Sixteen (2^4) response patterns are possible, ranging from ++++ to 0000. A χ^2 test enables one to estimate the latent structure from the observed data. Having obtained a latent structure which fits, one can compute an ordering ratio, which is the probability that a document having a given word pattern belongs to a particular latent class. For example, a document with all four key words present has a probability of .998 of belonging to class 1, i.e., it is concerned with computer automated instruction.

Table 3 shows the relationships between the response pattern, expected frequencies, and ordering ratios of the 1000 documents analyzed, in terms of their latent class structure.

Response Pattern	Expected Frequency		Total Fitted	Ordering Ratios	
	Class 1	Class 2		Class 1	Class 2
++++	158.76	.24	159.00	.998	.002
+++0	105.84	2.16	108.00	.980	.020
++0+	68.04	.96	69.00	.986	.014
+0++	68.04	2.16	70.20	.969	.031
0+++	17.64	.56	18.20	.969	.031
++00	45.36	8.78	54.14	.838	.162
+0+0	45.36	19.44	64.80	.700	.300
0++0	11.76	5.04	16.80	.700	.300
+00+	29.16	8.60	37.76	.772	.218
0+0+	7.56	2.39	9.95	.768	.232
00++	7.56	5.04	12.60	.600	.400
+000	19.44	77.76	97.20	.200	.800
0+00	5.04	20.16	25.20	.200	.800
00+0	5.04	45.36	50.40	.100	.900
000+	3.32	20.16	23.48	.142	.858
0000	2.16	181.20	183.36	.012	.988

Table 3. Expected Frequency of Response and the Ordering Ratios Based Upon the Estimated Latent Structure[1]

The table readily reveals the applicability of latent class analysis for information retrieval. This application is still in the theoretical and experimental stages. It has yet to be tested with empirical data from actual files.

AUTOMATED DOCUMENT CLASSIFICATION

The preceding discussions of factor analysis, clump theory, and latent class analysis all dealt with methods for devising empirically based

classification categories. These and other researchers have been investigating mathematical methods for deriving classification categories because of their belief that empirical classification systems will provide a more efficient means for the classification and the retrieval of information than the traditional methods of document classification. This belief has been subjected to scientific tests and evaluations.

In a study by Borko and Bernick,[3] an attempt was made to test the hypothesis that a classification system derived by factor analysis provides the best possible basis for automatic document classification and would result in more accurate automatic classification of documents than would be possible using more traditional classification categories. Maron,[7] in pursuing his interests in automatic indexing and classification, worked with 405 abstracts of computer literature which had been published in the *IRE Transactions on Electronic Computers*, Volume EC-8. In essence, Maron proposed a set of 32 subject categories which he felt were logically descriptive of the computer abstracts. Then he selected 90 clue words in such a manner that they would be good predictors of his 32 categories. The 405 documents were divided into two groups—260 abstracts made up the experimental group and the remaining 145 comprised the validation group. Maron classified all 405 documents into the 32 categories. Working with the documents of the experimental group only, he computed the value of the terms in the Bayesian prediction equations. He then used this formula to automatically classify the documents into their categories. Automatic document classification was correct in 84.5% of the cases in the experimental group and in 51.8% of the cases in the validation group.

Borko and Bernick decided to test the hypothesis that a higher percentage of correct classifications could be made using the same set of documents if a factor-analytically derived classification system were used instead of Maron's logically derived categories. However, their results were approximately the same as those obtained by Maron. Because of the nature of the experimental design used, it was impossible to determine whether the difficulty lay in the mathematically derived classification system or whether the factor score method used to predict correct document classification was not as effective as the Bayesian prediction equation.

Another series of experiments were designed and executed.[4] It was concluded from this series that, while there was no significant difference between the predictive efficiency of Bayesian and factor score methods, automatic document classification is enhanced by the use of a factor-analytically derived classification schedule. Approximately 55% of the documents were automatically and correctly classified. While this 55% current automatic classification of the documents is statistically very significant, it will have little practical significance until greater accuracy can be demonstrated.

Up to this point the criterion for correct classification has been the human classifier, but this is not necessarily the best criterion. We know that humans are not perfectly reliable, and therefore, it is not possible to predict human classification with perfect accuracy. The ultimate criterion of the usefulness of any indexing and classification system is whether it retrieves relevant information in response to a search request. Automatic document classification procedures should be evaluated on how efficiently they retrieve information and not on how well they can match the imperfect human classifier. This is a much more difficult problem, but research is already under way to evaluate the retrieval effectiveness of automatic document classification. As work progresses on the evaluation and improvement of techniques for automatic document indexing and classification, it can be anticipated that the bottleneck which now exists between the collection and the processing of documents will be eliminated and automated storage and retrieval systems will become possible.

BIBLIOGRAPHY

1. BAKER, F. B. Information Retrieval Based Upon Latent Class Analysis. *Journal of the Association of Computing Machinery*, Vol. 9, No. 4, Oct. 1962, 512–521.
2. BORKO, H. The Construction of an Empirically Based Mathematically Derived Classification System. *Proceedings of the*

Spring Joint Computer Conference, San Francisco, May 1–3, 1962, Vol. 21, 279–289. (Also available as SDC document SP–585.)

3. BORKO, H., and BERNICK, M. Automatic Document Classification. *Journal of the Association of Computing Machinery,* Vol. 10, No. 2, April 1963, 151–162. (Also available as SDC document TM–771.)
4. BORKO, H., and BERNICK, M. Automatic Document Classification: Part II—Additional Experiments. *Journal of the Association of Computing Machinery,* Vol II, No. 2, April 1964. (Also available as TM–771/001/00.)
5. BRITISH STANDARDS INSTITUTE. *Guide to the Universal Decimal Classification (UDC).* British Standards House, London, 1963.
6. HARMAN, H. H. *Modern Factor Analysis.* University of Chicago Press, Chicago, 1960.
7. MARON, M. E. Automatic Indexing: An Experimental Inquiry. *Journal of the Association of Computing Machinery,* Vol. 8, No. 3, July 1961, 407–417.
8. NEEDHAM, R. M. The Theory of Clumps, II. M. L. 139, Cambridge Language Research Unit, Cambridge, England, March 1961.
9. OLNEY, J. C. FEAT, An Inventory Program for Information Retrieval. SDC document FN–4018, July 1960.
10. PARKER-RHODES, A. F. Contributions to the Theory of Clumps, M. L. 138, Cambridge Language Research Unit, Cambridge, England, March 1961.
11. SHERA, J. H. and EGAN, M. E. *The Classified Catalog: Basic Principles and Practices,* American Library Association, Chicago, 1960.
12. TANIMOTO, T. T. An Elementary Mathematical Theory of Classification and Prediction, IBM, New York, 1958.

INFORMATION STORAGE AND RETRIEVAL– ANALYSIS OF THE STATE OF THE ART

G. N. Arnovick, J. A. Liles, and J. S. Wood

Information Storage and Retrieval Systems
Systems Engineering
Space and Information Systems Division
North American Aviation, Inc.

INTRODUCTION

Information retrieval, like the weather, stimulates a good deal of verbal clamor and speculation, yet remains vexingly elusive and unmanageable. Hopefully, both the weather and recorded information will eventually prove amenable to some form of human control. In the meantime, there must be a continuing effort to achieve balance in the evolution of the concept and equipment aspects of information storage and retrieval (IS&R). It is reasonable to expect overemphasis on equipment capabilities. The emergence of IS&R as a distinct discipline is largely attributable to the significant advances of modern computer technology.

Nevertheless, steps must be taken to correct the deficit in systems and concepts, or the situation will be analogous to a surveyor pacing off chains with a precision micrometer.

The purpose of this review is to assess present capabilities in the field and the extent to which these capabilities are effectively utilized. However, in view of the clamor that has already been made about IS&R, the inclination here is to avoid a massive item-by-item listing of available systems and techniques. Instead, general classes of the operations and equipment involved in IS&R are summarized and evaluated. Specific examples are cited for illustration.

In anticipation of the rather broad category of readers to whom this paper is directed, an attempt is made to reasonably satisfy those with considerable background in IS&R as well as those with less familiarity. Consequently, an attempt is made to avoid laborious and detailed descriptions of coordinate indexing, storage media, etc. At the same time, it is recognized that some readers may, for example, be technically expert in hardware but require some familiarization with the specific software notions of IS&R. It is emphasized, therefore, that the central aim is to present a critique rather than to educate or merely inform.

These considerations necessitate inclusion of the somewhat fundamental (though cursory) earlier sections on hierarchic and coordinate indexing. They should be ignored or quickly scanned by the informed reader. Those who have been moderately exposed to the subject should be interested in the sections on Problems of Coordinate Indexing and, to a lesser extent, on Relationships in Coordinate Indexing. The descriptions and evaluations of probabilistic and automatic indexing should be useful even to those with extensive documentation background.

The greater portion of the review is devoted to a discussion of indexing for this is considered to be the most important single factor in IS&R.

For example, it is impossible to embark upon an exposition of coordinate indexing without touching on the coordination problems that develop at the retrieval end ("false drops" as one instance). Considerable space in the section on indexing is in fact dedicated to the problem of coordination failures.

The last part of the section on indexing touches lightly on the subject of storage media to convey a feeling for the ways in which indexes are made readily available to the browsing inquirer.

The remaining aspects of information handling—storage and retrieval, abstracting, dissemination, reproduction and display, and communication links—are treated more summarily. Conclusions are presented at the close of the review.

INDEXING

The schemes for indexing documents are multifarious and growing. There are any number of ways in which all the existing and presently conceivable schemes may be organized, but the plan in Figure 1 seems convenient for obviating some of the ambiguities and redundancies usually encountered.

Kent[19] gives the following definitions of document and aspect systems:

> Document systems may be defined as those information systems that involve the recording of all characteristics concerning a single document on one record or, less commonly, on a single discrete set of records.
>
> Aspect systems may be defined as those information systems that involve the recording,

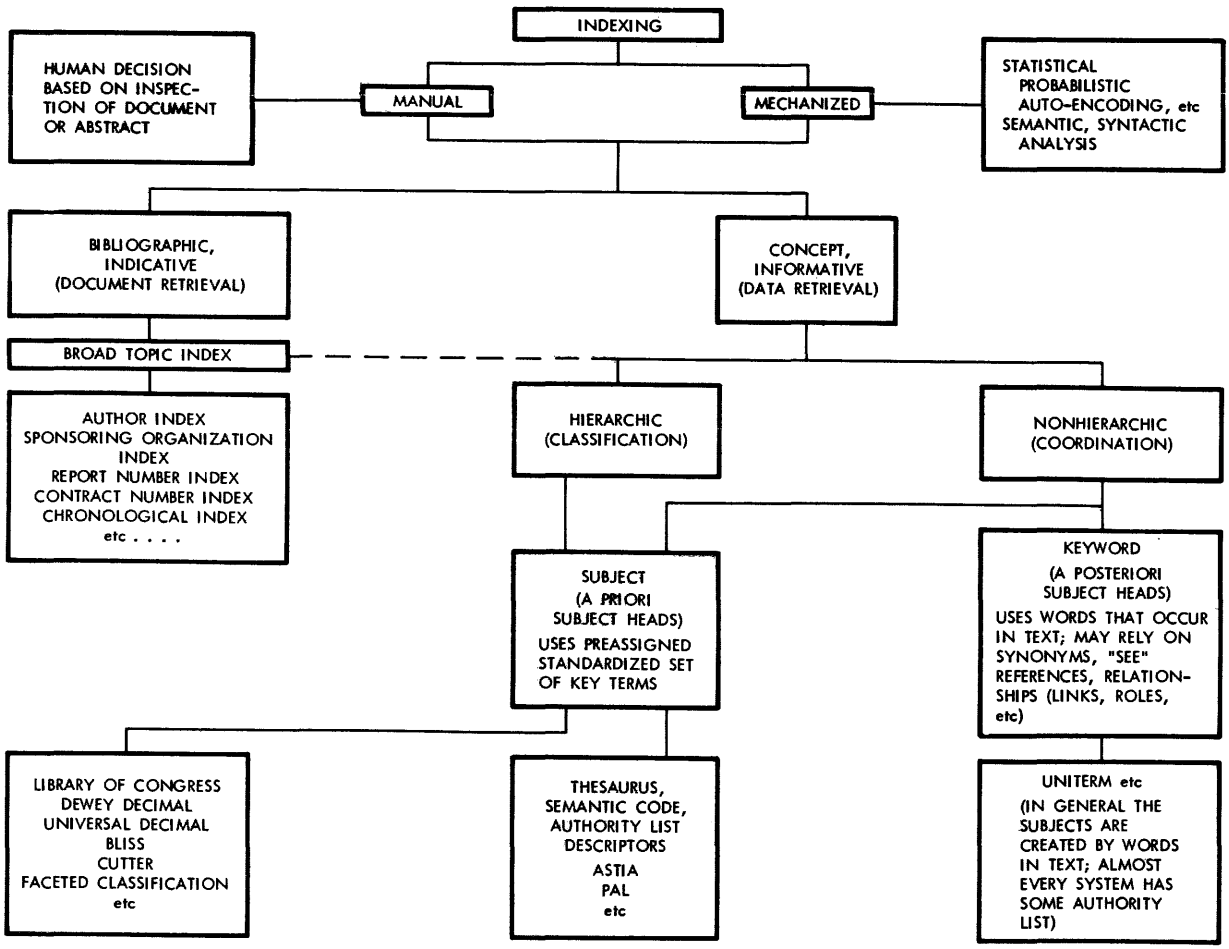


Figure 1. Breakdown of Indexing Schemes.

on one record or a discrete set of records, of the numbers (or other unique identification) of all documents that have a characteristic (or aspect) in common.

In general, indexing systems that locate specific information are characterized by greater depth of indexing than systems that merely locate documents. But increasing the depth amplifies the concomitant problems by orders of magnitude. Indicative indexing is a working reality; information indexing, at least where a large general store of information is concerned, is a monster whose powerful potential needs to be harnessed.

Almost all present working systems still rely on human indexers, even if all other aspects are automated.

CONVENTIONAL SUBJECT CLASSIFICATION

In conventional subject classification, all documents and books are filed in a given slot according to an established, predetermined hierarchy of subjects. This system is poorly suited to a modern technical library for the following reasons:

Lookup is a two-step operation involving a search for documents under a subject head followed by a search of subjects under each document to choose those which are pertinent.

Overlapping of disciplines is increasing.

The process of interpolating new terms (updating) in a rapidly developing subject is cumbersome.

A conventional universal classification scheme may include the entire store of an aerospace-oriented library in one small corner of its structure, whereas aerospace may span a colossal and varied collection of highly overlapping topics.

Most hierarchies are artificial. The natural structure of biology and chemistry does not extend to all knowledge. Extensive cross referencing alleviates the confinement of one heading to a reference, but it burdens the user and, carried to extremes, renders chaotic the already synthetic hierarchies.

FACETED SUBJECT CLASSIFICATION

Faceted classifications attempt to introduce greater flexibility by permitting the free combination of various categories or facets. Subjects are not arranged in a fixed descending order but are combined in an order prescribed according to classes. The chief advantage in addition to flexibility is that the classification may be adapted to the information that is being classified.

Nevertheless, a document is permanently pigeonholed, and most of the inherent disadvantages of subject classification are still present.

In fact, faceted classification fails in general because it does not solve the problems of the searcher. If the hierarchy is not a natural one (and most are not), it represents a point of view. This system is suitable for well-established subjects and, hence, for the filing of certain (e.g., reference) books. Enough time has elapsed for the user to have become familiar with the indexing point of view. However, the outlook of the searcher and the order of the index may be completely disparate in considering a current, complex subject.

HIERARCHIES AND RETRIEVAL EFFICIENCY

To better understand the problem of classifying knowledge in rapidly evolving fields, the following example is cited. Even if retrieval is 100-percent effective, problems still remain. Herner and Herner[16] found in a recent study involving atomic energy materials that the coincidence between the content of research reports and reference questions in atomic energy is very small. Further investigation revealed that once reports have been used as sources of current information, their significance falls away rapidly. These results powerfully emphasize the fact that the time available for matching viewpoints is essentially zero.

COORDINATE INDEXING

A natural consequence of the development of high-speed computers was the attempt to find new ways of processing data including documented information in machine-tractable form. One reasonable approach was to question the

continuing necessity for creating artificial relationships between subjects as in hierarchic classifications. Every document defined its own subject. How then could the relationships between concepts and data be determined by the actual environment in which they were embedded?

The discussion that follows treats the concept of coordination as originally conceived. It presupposes few or no a priori associations between concepts. Certain significant words or data are taken from the text of a document, and the searcher is then free to combine any number of such key terms as he chooses in the attempt to match, not viewpoints, but what he actually has in mind with what the author is actually talking about. This approach is the ideal. The ways in which it can be implemented and the extent to which the ideal is approached in practice need to be examined. The methods of implementation and storage techniques and will be discussed under that subject.

The ideal is compromised when associations are made between index terms. In a recent issue of *American Documentation*, it is argued that the introduction of relationships is tantamount to denial of Uniterm indexing and return to subject heading classification (attributed to Cutter in 1876).[2] Certainly lexical and grammatical ties play an important part in information content. But considering the expense and state of the art of linguistic analysis,[4] the attitude is that coordinate indexing (no relationships) and subject heading classification (inflexible with respect to association) are at opposite poles; something flexible and capable of expressing relationship lies between.

The general concept of coordinate indexing is well understood and will be sketched only briefly; it involves three basic steps:

1. Accession numbers are assigned to the documents in the store. Consider 10 documents lettered A to J comprising a closed store.
2. From each document, the indexer picks out a representative number of word-tokens (single occurrences of a word in a document) and assigns a code number to each word (a given word distinct from any other; it may occur in several documents). Assuming emergence of a total vocabulary of 30 numbered words, the result may be depicted as shown in Tables 1 and 2.

Table 1. Documents Ordered by Term Number (Serial List)

Document Number	A	B	C	D	E	F	G	H	I	J
Words	①	3	5	3	16	8	21	7	11	28
	2	6	7	12	17	11	22	①	8	4
	3	4	①	13	2	19	23	8	2	29
	4		8	14	18	20	13	24	①	30
	5		9	9		15			25	①
			10	15		18			26	
			11						19	
									27	

In this listing, the 30 words are represented by 52 word tokens.

3. The list is inverted by grouping all document accession numbers under a given word. For example, word number 1 (circled) occurs under document letters A, C, H, I, and J. The inverted list looks like this (for the first 10 words):

Table 2. Documents Ordered by Term Number (Inverted List)

Word Number	1	2	3	4	5	6	7	8	9	10
Document Number	A	A	A	A	A	B	Ⓒ	Ⓒ	C	C
	Ⓒ	E	B	B	C		Ⓗ	Ⓗ	D	
	Ⓗ	I	D	J				F		
	I							I		
	J									

A search defined in terms of words 1, 7, and would be found in documents C and H, which have these numbers in common.

Problems of Coordinate Indexing

Indexer Variability

The practical difficulties incurred by straight coordinate indexing arise primarily because the effect of viewpoint can never be eliminated. It will occur wherever decisions are made, and

indexing at the present stage is largely a decision-making process.

In a superficial experiment conducted at Space and Information Systems Division of North American Avaition, the IS&R group selected a series of documents and compared the indexes produced by different people (none of whom, however, was a trained indexer). The variations were enormous. Without attempting to generalize the few results of this test, the following trends were noted:

Those who indexed documents within their own specialization tended to read (rather than scan) the document and pick many more than the average number of terms.

Those with technical background who had not specialized in the field tended to be more selective and pick terms that seemed to be emphasized either by frequency of occurrence, placement in quotes, or other attention-drawing devices of the author.

Nontechnical people tended to pick a fair number of terms, chiefly selecting those which looked "technical" and often ignoring "common" terms with a meaning specialized in the particular documents.

Certain individuals consistently pick few terms, others pick many. Citing the most extreme example, a document on mathematics was indexed by one person (a mathematician) with 52 terms and by another (an aeronautical engineer) with 5 terms.

There was no effort to evaluate document relevancy as a function of the resultant indexes.

Extensive formal studies of this kind are being made at Documentation Incorporated (indexing reliability tests and study of teaching/learning aids)[6] and IBM (study of effect of educational background on encoding performance).[31] Savage reports no significant effect of educational level on ability to index.[18]

Clearly, the human factors in manual indexing need to be explored more fully before effective automation of the process can be achieved. Specifically, indexing techniques and term selection need to be assessed in terms of retrieval effectiveness.

Search Variability—Coordination Failures

A second problem is the variability with which the searcher (or requester) can express his inquiry. Often it happens that he is unable to express it. A document on germanium rectifiers might be overlooked in a search for germanium diodes (near-synonym) or for germanium transistors (inclusive class). Conceivably, a requester might even verbalize his inquiry as "a germanium thing that permits current to flow in one direction but not in the other."

This problem is one of a group that is categorized as coordination failures. These are best portrayed by example. Returning to the set of documents, A to J, and their filial terms, 1 to 30 (Table 1), the following specific topics and words are assigned:

Document E
on: sodium chloride, potassium chloride, and potassium iodide

Document G
on: air-to-ground missiles

Word no.:	2—potassium
	16—sodium
	17—chloride
	18—iodide
	21—air
	22—ground
	23—missiles

Table 3 lists the types of coordination failure with examples of searches in which they might occur. The examples are purposely trivial for the sake of illustration.

Relationships in Coordinate Indexing

There are some purists who feel that any presumption of relationship between index terms is flatly a regression to the construction of hierarchies with all of their concomitant

vexations, subjectivity, and artificiality. Others insist that there must be an ideal marriage of coordination and classification to produce optimum communication between the information store and its user. Sensibility lies with the middle group that comprises not only the greatest number of documentation practitioners (those actually engaged in meeting the information needs of scientists and engineers) but also some of the most advanced thinkers in the field.

Table 3. Examples of Coordination Failures Using Pure Coordinate Indexing (Selection and Juxtaposition Only of Terms Occurring in Text)

Type of Coordination Failure	Example Request	Search Terms	Documents Retrieved	Reason for Failure
1. False coordination a)	Ground handling of missiles	Ground (22) Missiles (23)	G (air-to-ground missiles)	Insufficient depth of request—easily rectifiable by narrowing specificity of request.
b)	Sodium iodide	Sodium (16) Iodide (18)	E (sodium and potassium chloride, potassium iodide, but NOT sodium chloride)	Need to show stronger relationship than mere co-occurrence or juxtaposition. Possible solutions (links, roles, etc.) are discussed under relationships.
2. Incomplete coordination	Air-to-air missiles	Air (21) Missiles (23)	G (air-to-ground missiles)	The search terms are sufficient to describe the request but insufficient to describe document G, which is nevertheless retrieved. A meager solution is the use of logical negation (air AND NOT ground), but the requester or searcher must be aware of possible exclusions.
3. Synonym failure	Table salt	Table salt	NONE, but document E is relevant	Index does not provide for synonym "table salt-sodium chloride." A thesaurus or subject authority list is needed.
4. Generic search failure	All documents on alkali halides	Alkali halides	NONE, but document E is relevant	Index does not provide for generic name of chemical compounds, i.e., "alkali halides NaCl, KCl, KI." A thesaurus or authority list is required.
5. False relationship	Ground-to-air missiles	Ground (22) Air (21)	G (air-to-ground missiles)	Exact matching of terms (i.e., no absent terms, no superfluous terms), but a stronger relationship than mere co-occurrence or juxtaposition must be shown.

Table 3. Examples of Coordination Failures Using Pure Coordinate Indexing (Selection and Juxtaposition Only of Terms Occurring in Text) (Cont.)

Type of Coordination Failure	Example Request	Search Terms	Documents Retrieved	Reason for Failure
6. Failure to retrieve "next best" documents a)	(Refer to Table 1)	3, 6, 4, 31, 32	NONE. The requester, however, might find it advantageous to examine document B, which has three of the search terms (3, 4, 6)	No facility for pulling documents with at least some of the search terms (optimally, the most terms); lack of browsing capability.
b)	(Refer to Table 1)	3, 31, 32	still NONE	Same problem as above; merely shrinking the request does not necessarily improve the retrieval. If a searcher began a process of successive elimination of terms, in the given example he could make as many as $\left(\frac{5}{1}\right)+\left(\frac{5}{2}\right)=15$ attempts before getting a "hit." This would only be practical on computers and even then is not the most efficient approach.

Physical Proximity

A primitive kind of association between words in a text is the distance in terms of words that separates them. This association is thought by some to be useful in machine translation as a word and its dependents tend to be grouped together, but it seems to have little application to indexing. It could serve a useful analytical function if index terms were tagged according to the part of a document in which they occurred, such as title, abstract, and main body. The relative importance of these parts could then be analyzed. A related study has been made by workers at IBM. Resnick[30] and Savage report that the results of indexing from an abstract and from the title of a document are not significantly different. It should be realized, however, that the IBM indexing studies are based on a comparison of indexing terms with keywords supplied by users of a Selective Dissemination of Information (SDI) System. A certain bias is thus already inherent in the tests.

Synonyms

Words with similar meanings can be handled in the input by using "see" references. In a descriptor system, all words considered near enough in meaning are mapped into a single term (descriptor) via a subject authority list or they are referred to a thesaurus. The process can be done by the human searchers or, prefer-

ably, can be handled entirely within the system dictionary. If the system is to serve its users with a minimum of inconvenience, the constraints imposed on the users and indexers must be reduced to a minimum. Humans should be free to make decisions (formulate requests, select keywords). The system should unburden them of the processing and matching operations.

Partial Implication or Near-Synonyms

This problem is much like the preceding one except that now provision is made for generic groupings of specific terms and specific filials of generic terms. These are usually handled by "see also" references. In other words, some documents are filed under the given term but other potentially relevant documents may be found under related terms. This technique incorporates into the system the main advantages of hierarchic arrangements.

Semantic Relationships

As a rule, the association of index terms should not be stressed too heavily. It is sometimes helpful to show directional relationships in answer to the question "Does A affect B, or does B affect A?" For example, the terms "diffraction" and "ultrasonic" could pertain either to ultrasound used to diffract light or to the diffraction of ultrasound. Even the expression "utrasonic diffraction grating" can be ambiguous in this respect.

The occurrence now and then of such false or ambiguous correlations led to the creation of associative links that connect various terms from a document in one group, other terms from the document in another group, and so on. To further clarify the meanings involved, semantic roles were devised to delineate the function of a given term within a given document.[12] The foregoing ultrasonic diffraction grating problem could be handled nicely by assigning "ultrasonic" a role indicator telling whether it is "acted upon" (by an acoustic grating) or "acting upon something" (light).

A present working system that uses roles and links extensively is the one serving the American Institute of Chemical Engineers.[26] However, such devices are best suited to such highly structured disciplines as chemistry since it is important for the chemist if he is interested in a given chemical as a catalyst not to be bothered with documents on this chemical as a reagent, dye indicator, or insecticide.

Another device is to join words inseparably as bounds terms (precoordination). A noun and its descriptive adjective might be treated as a single concept. This has both good and bad consequences. It requires a well-tutored indexer and means that a search for one of the terms will not, in general, pull a document indexed with the bound term. In particular, however, it is useful for concepts such as chemical compounds (especially in a chemical library) or the occasional ambiguous term like air-to-ground missiles. It is beneficial for pairs of terms that always coordinate.

Relationships between words are invoked in the hope of decreasing false coordinations. But an exaggeration of their importance is not desirable, for they can simultaneously exclude relevant documents. A good system will combine the various types of coordinate indexing, avoiding extremes but using relationships sparingly.

Probabilistic Indexing

Indexing, like searching, may be thought of as a binary operation. Every word in the text of a document is either chosen as an index term or it is not. In a search, a document is either picked (assumed relevant) or it is not. This means that when a request is made, the user has reason to be wary of the output. He may get a handful of documents, most of which will hopefully be relevant to his request. Or he may be handed a list of several hundred accession numbers. He may then undertake the project of scanning the documents or narrow his request. Then he may exclude documents of interest. All this because the system called every document either good or no good.

Probabilistic indexing, proposed by Maron and Kuhns,[24] provides for the assignment of weights to index terms and the ranking of retrieved documents according to some criterion of relevance. Instead of being chosen on a zero-one, go or no-go, basis, the words of a document are weighted by the indexer according to their significance to the documents. Maron and Kuhns, in their own experimental work, use an

eight-level rating, i.e., 0, 1/8, 2/8, ... 7/8, 1. Relationships are also introduced, but on a purely statistical basis relying on the frequency of occurrence of word pairs in all the documents of the collection. The most promising one of several mathematical devices used to relate the word pairs is a coefficient of association.

The coefficient of association is, as described by the authors, essentially a measure of the excess of joint occurrences of a given pair over expectation based on random co-occurrence.

An alternative to Maron's "excess" was presented by Stiles at the recent symposium on materials information retrieval attended by one of the authors.[35] The chi-square test used to correlate physical experimental data is applied to word co-occurrences.

Probabilistic Indexing Experiment

A controlled experiment was conducted with 110 articles, selecting and categorizing keywords then working backward to coordinate documents and categories. In this way, eventually, for every request there was an answer document which, when retrieved, satisfied that request. Conventional (binary) and probabilistic techniques were compared, and the results were expressed in terms of the number of retrieved documents that had to be read before hitting the answer documents.

It turned out that the conventional system would require the user to read approximately thirty percent more retrieved documents to obtain the same number of answer documents as opposed to the basic selection process of probabilistic indexing. Considerable improvement was achieved with elaboration of the basic selection process.

Probabilistic Indexing—Critique

Indexer subjectivity is greatly increased. The decision-making process would seem more difficult, but it may subsequently prove to be worth it. According to the authors, the work of the indexer is facilitated; as it now stands, he is too constrained by having to say yes or no to a possible index term.

The ability to make too fine a breakdown of the weighting scale from 0 to 1 is questioned. An analogous problem arises in education when a teacher has to grade pupils in a highly subjective field. In probabilistic indexing a scale of 0 to 3 (or 0, 1/3, 2/3, 1) would seem to be the maximum. However, the psychometric problem of scaling is not an easy one to resolve.

The premise that one can go from index terms that are highly significant of a document to documents that are highly relevant to a request is accepted with much reservation. It is probably a correct premise in most instances, but it overlooks the situation in which a request will be quite well and perhaps best satisfied by a document in whose over-all text the search terms play a relatively minor role.

The old subject-classification problem—to describe the subject of a document—does not wholly describe its information content.

There are many formulations of the information retrieval problem in terms of statistical and information theory, but most of them skirt the problem. In fact, the circumstances are often tantamount to a search for the exact solution to a problem when most people are not sure just what the problem is.

The probabilistic indexing approach, on the other hand, has inherent potentialities for the eventual over-all mechanization of IS&R. This, Maron's later work on automatic indexing and Edmundson and Wyllys' insight into the true implication of information theory for IS&R, seem to be paths to the library of the future.

Probabilistic indexing is still a manual indexing procedure. It appears to be the natural transition from manual to automatic indexing because as mechanized statistical methods of selecting terms from a document come more and more into use, probabilistic indexing will provide a richer foundation on which to build.

MECHANICAL INDEXING

Any discussion of mechanized indexing should be prefaced by citing the single major limitation that prevents any of the elegantly devised theoretical schemes from being put into practice. This is the requirement for total text input. The machine processing of total text is not efficient for anything but experimental analysis as long as printed text must be translated into machine-readable text (via keypunching, for example). Working automatic

indexing systems, like machine translation systems, must await the perfection of print-reading and character-recognition devices.

Mechanized indexing is attacked from two different angles, statistically and linguistically.

Statistical

The common basic assumption of statistical selection processes is that the linguistic properties and laws of literature are describable in terms of populations and frequencies of word occurrences and their associations. The difference between statistical description of language and statistical thermodynamics is basically quantitative, but this quantitative difference is sufficient to manifest itself in a qualitative form. In general, the numbers treated by thermodynamics are larger and the variables fewer than in linguistics. The result is that given a norm of behavior in linguistics, the deviations from the norm are more pronounced than in thermodynamics. There is little evidence in the physical world of less than maximally probable events. In language, on the other hand, exceptional occurrences are witnessed regularly.

The problem confronting mathematical linguistics, then, is to define the variables and to refine the statistical behavior laws narrowing their specification and reducing the undefinable to a minimum.

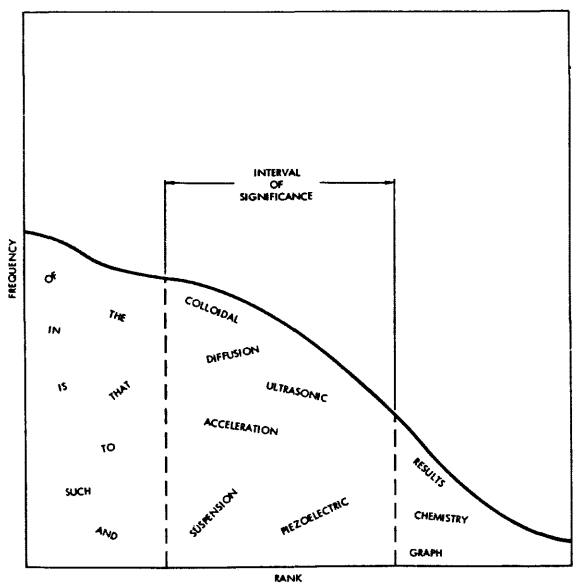


Figure 2. Idealized Frequency Count of Words Occurring in a Document Entitled "The Ultrasonic Acceleration of Diffusion."

The fundamental concept of statistical key word selection is illustrated in Figure 2, which is designed to convey a graphic impression of the concept and is not necessarily the only procedure for implementing the concept. The diagram represents a smoothed frequency count of the word occurrences in a document as a function of the rank order of the words. This is statistical selection in its crudest, most unrefined form. It flatly presupposes that common words will occur most frequently, the esoteric but unpertinent words most infrequently. The middle-frequency words—those remaining after the extremes of the curve have been removed—are considered to be the most significant. The same process can then be applied to pairs of the significant words.

In general, this first-approximation procedure screens out the many clearly irrelevant words leaving a working body that can be treated by more precise techniques.

Examples of Statistical Selection

Three proposals for automatic, statistically based indexing merit attention. One has many ramifications in use today; the other two are still in the ivory tower having been subjected only to carefully controlled experiments.

Auto-encoding

The pioneer study of mechanized indexing,[22,23] which led to the key word in context (KWIC) index, relies on the computer recognition of individual words and counts their frequency in a text. Eliminating the common words, the most commonly occurring topical words are used (without further discrimination) as index terms. The product is a mechanically prepared concordance.

Recognizing the limitations of word indexing, Luhn has standardized the vocabulary by combining words with the same root and then combining the counts of synonymous words. A normalized form is selected after being looked up in a thesaurus. Relationships are handled by computer analysis of word pairs in cases where significant words occur together.

The search terms are developed analogously from an essay-form request by matching search terms and index entries.

Automatic Document Classification

Academic-level arguments to the contrary notwithstanding, there is much to be gained by a scheme for rapidly and automatically fitting documents into sensible categories. The problem facing library practitioners is particularly acute in this respect, because while those slightly removed from the bookshelves debate the comparative merits of classification versus document-independent organization of information, these people are pressed to shelve their documents in a rational fashion. Besides, as Borko argued so convincingly at a recent linguistic seminar,[6] people (scientists included) tend to think naturally and with facility in such an ordered way, i.e., categorically. The conflicts arise when the flexible adaptive human psychological system confronts rigid, long-standing systems solidified on paper.

A happy solution appears forthcoming in automatic classification. Work has been done in this area by Maron[25] and Borko et al.[5,6] In both instances the researchers rely on statistical correlation between words and documents, and between words. The two experiments are similar in several respects. For example, documents are searched by computer for the occurrence of specific clue words relating to specific a priori subject heads, or categories. The essential difference lies in the derivation of the categories and assignation of the documents. Maron devises his own categories and assigns the documents by means of a Bayesian prediction formula. Borko, on the other hand, derives his categories by matrix correlation of the clue words, extraction of eigenvectors by factor analysis, and rotation of these eigenvectors to find the best categories. Documents are then assigned by factor score prediction techniques.

Borko has done considerable work in the last couple of years in comparing his own and Maron's work. A related effort in document association by linguistic analysis has been undertaken by Salton at Harvard.[33]

Automatic Indexing (Maron)

Maron's scheme[25] for automatic indexing operates by generating a priori subject heads and searching documents by computer for the occurrence of these subject heads (called clue words). The intention is a good one, to ultimately merge probabilistic and automatic indexing.

Automatic Indexing (Edmundson and Wyllys) —Relative Frequency Approach

This technique[10] represents a new concept in the statistical analysis of text and comes closest to bridging the gap between IS&R and information theory. Instead of treating a document as the universe of words as in the primitive diagram in Figure 2, the frequency of a word in a document is compared with the frequency of the same word in general use. Various statistical criteria are suggested and evaluated. The upshot is that a word that is rare in general use but frequent in a document is a significant measure of that document's content. This corresponds to the surprise element of information theory.

Citation Indexing

Several studies have been made in recent years using the bibliographic materials and citations contained in documents either directly for cross-referencing purposes or as a statistical vehicle.[12,13,20,33] These techniques exploit the dynamical historical evolution of science and the language used to express it. One of the authors (Wood) is working on interlingual citation statistics in Russian and English acoustics journals in an attempt to derive systematically from current terminology an accurate technical terminology for machine translation and information retrieval from foreign language references.

Linguistic

The creation of indexes by automatic linguistic analysis penetrates deeply into the realm of machine translation (MT). In this case, it calls for the automatic translation of the source language into a target IS&R language from which indexes can be created. Present MT efforts are aimed at syntactic analysis, of which there are two types:

1. Immediate-constituent or phrase-structure analysis—the segmentation of sentences into successively smaller machine-tractable parts.

2. Dependency analysis—the assignment of a governing word to every word in a sentence (or other lexical unit), i.e., the creation of a complete dependency structure.

Further elaboration of linguistic analysis is beyond the scope of the present survey. A clear description and reconciliation of the two approaches are given by Hays.[15] Klein and Simmons actually implement both procedures simultaneously.[21] Finally, a thorough state-of-the-art treatment of syntactical analysis was presented by Bobrow at the last Joint Computer Conference.[4]

Extension of Concepts

A desirable future system would unify automatic statistical indexing applying the relative-frequency approach of Edmundson and Wyllys as a criterion for assigning weights to index terms as required in probabilistic indexing.

A certain amount of skepticism is evoked by the assumption that statistical selection will create the best indexes because some of the problems are semantic. However, it will probably create the best automatic indexes for some time to come.

Linguistic analysis has a serious drawback if used to find a common language for IS&R and MT in that there is a major difference of goals between IS&R and MT. In MT, the user supposedly has a document, wants to know what it says, and is therefore interested in fidelity of translation. In IS&R, the user is interested in finding a peculiar (sic) type of information and wants the documents that contain it regardless of how it is presented. On the other hand, linguistic analysis will serve IS&R advantageously if used to learn how people think and communicate ideas.

An intermediate step to probabilistic indexing would be indexing on two levels, viz., terms which are indicative of what a document is about and other terms pertaining to items of information not immediately contained in the topic. In other words, the following assignation of values would be made to every word in the document:

0 — Nonsignificant word
1 — Significant nonindicative word
2 — Significant indicative word

Indicative is interpreted to mean that the word is directly related to the theme of the document. Such distinctions are psychologically facile for human indexers.

STORAGE AND RETRIEVAL

PRESENTATION OF INDEXES

If an automatic literature search is conducted, the user will, in general, never see the index. He will make a request, and the system (via librarian or computer) will do the search for him.

There still remains a strong need for the user, on occasion, to see the index. The forms of index presentation, all of which can be readily produced by machine, are card form, book form, and pictorial display.

Card Form

Every library user is familiar with the standard card catalog. The cards may represent units of a document file, in which case one card contains all the printed information pertinent to a document. They may be units of an aspect system, each card representing an index term and containing a list of document numbers. They can also be punched or otherwise marked for manual or machine searching, and they may have a window with a microfilm of the document or abstract (aperture card).

Book Form

Conventional and Rotated

Book-form indexes stem from the familiar list of topics at the back of almost every nonfiction book. The frustration often engendered by most book indexes led to improvements, such as the extremely detailed and cross-referenced index of Chemical Abstracts. An advanced development is the rotated, or permuted, index. Strings of key words, such as the significant words of titles or auto-abstracted sentences (KWIC), are listed several times, alphabetically according to each word in every such string. Many experimental machine-generated index printouts of this type are now in use and are proving to be very effective. For example, the RotaForm Index, now a regular part of Index Chemicus, is a computer-generated molecular formula index that repeats and rotates

a chemical formula for every chemical element that it contains.

Columnar

An attempt to present coordinate indexes of the uniterm type in book form is the columnar index, such as the Scan-Column Index and Tabledex. These indexes print adjacent columns of document numbers under word numbers and are rapidly scanned to locate documents with a large number of desired key words. The limitations for an untrained user are pretty obvious. Clearly, the fatigue factor would be high.

The book-form index is, of course, a closed index; once it is printed, terms cannot be entered or deleted.

Pictorial Display

The ability to visualize a coordinate index in Gestalt form is much to be desired. It has the added advantage that it can, in principle, be viewed on the same screen as the document or abstract itself.

Termatrex and Minimatrex

Conceivably, one might project any book-form index on a screen. But the displayed index used to great advantage by a number of firms today is the Termatrex system. Document accession numbers are located on the nodes of a coordinate grid that is set against an illuminated background. When a number of Termatrex cards, each one representing a key word, are superimposed, the light will show through the nodes that are common to all of the key words. In this way, the user can get a rapid over-all glimpse of how many documents he would have to inspect as well as their accession numbers.

Also under development by Jonker Business Machines, the creators of Termatrex, is Minimatrex, an improved, miniaturized, mechanizable version of Termatrex. The coordinate grids are photographed on film strips by grouping broad categories of terms onto corresponding frames of the strip. In its present form, the Minimatrex strips have 8 frames, each one a 100-by-100 matrix representing 10,000 documents. As many as 12 strips can be superimposed, and the resultant retrieval pattern is projected on a viewing screen. The number of frames and superimposable strips is expandable.

Hypothetical Systems—
The Semantic Road Map

There is one hypothesized system which, although far too advanced conceptually to be operational even in the near future, deserves special attention because it foresees the possibilities of a truly automated era. Loren Doyle of SDC[9] gives an intriguing picture of a space-age library system. Giving credit to the relative-frequency notion of Edmundson and Wyllys,[10] Doyle seeks to "increase the mental contact between the reader and the information stored so the reader can proceed unerringly and swiftly to identify and receive the message for which he is looking."

The semantic road map idea recognizes the human proclivity to organize concepts in spatial relationship to each other to allow the free evolution of concepts, one as the outgrowth of another, linking the retrieval process more intimately with the process of natural creative thought.

Doyle provides a scheme analogous to the one by which we would find an apartment number, given an address number, street, zone number, town, county, state, and country, none of which were familiar to us. We would begin with a reasonably compact map of the world and proceed until we had a houseplan of the apartment building.

The projected library user sits before a console with a panel of control buttons and a viewing screen. He begins with a map of the contents of the library whereon he views concepts whose mutual associations are indicated by proximity, heavy lines, thin lines, dotted lines, large letters, small letters, and arrows that point toward distant cousins. Through a rapid succession of such maps, he homes in on the message for which he seeks.

The documents themselves are in the form of "document proxies," or abstracts in map representation similar to the search maps. This permits rapid scanning of a large number of document abstracts to find those worth reading.

The maps are, of course, automatically generated. One set of techniques for realizing such

a system is given along with the description of a preliminary experiment.

Don Swanson,[38] Dean of the Graduate Library School, University of Chicago, gave a somewhat similar man-machine concept for a library request console tied into a storage and control system. However, his discussion, unlike Doyle's, was centered on console-library request techniques as a feedback link which is part of an on-line search process system for future automated library systems.

Savage emphasizes that to describe the effectiveness of a system in terms of magnitudes and system parameters (number of books in a library or number of descriptors in a computer memory) does not tell how well the system works. What is needed is a standard criterion of relevancy, one that is as useful as miles per gallon for measuring automobile economy.

Savage characterizes human evaluations in a somewhat more precise way than is usual, points to some of the difficulties of ever generating a metric measure of system performance, and suggests the direction of study to obtain more accurate measures.

It is important to realize that any metric criterion of adequacy will have to be tested by way of obtaining feedback from the users of a real system.

MECHANICAL SEARCHES

The original mechanized searches were made by serially operating machines, which operate most efficiently by reading the whole store to find what is needed. Serial searching is still advantageous when the store can be broken up and grouped (as in multiple-punched card decks); input, updating, and purging are also simpler then.

By its very nature, however, serial searching is less efficient than applying the lookup principle, which is feasible with random access machines having large memories. This principle also allows for indefinite expansion of a large file.

SERIAL AND RANDOM ACCESS

The factors which must be considered in a comparative evaluation of serial and random access techniques (search and lookup) are file size, length of search, economy of money and time, effort expended, and the inherent efficiencies of retrieving a given type of information.

In the face of the growing number of available computer systems, the thorough-going and detailed evaluation of any systems or combinations of equipment presents an enormous task. An interesting and original approach to the problem by automation of the selection of systems suitable to a particular need is given by Arnovick.[1] Essentially, a complete list of system component characteristics and ancillary information is generated and stored in a computer. A complete set of application requirements is compiled and matched against the system information stored in the computer; the computer analyzes the application information and synthesizes an optimum general system.

LOGICAL OPERATIONS

It is convenient to think of searches as comprising two kinds of logical operation, conjunctive and disjunctive. Searches are narrowed by the former because taking the logical products and differences of terms (AND, AND NOT) makes the request more specific. Disjunctive operations are useful for broadening the search; the logical sum (OR) makes for greater variability. Disjunctive operations and combinations of disjunction with conjunction (logical products of logical sums, etc.) are especially helpful for processing request terms in a machine-stored thesaurus or otherwise pinpointing a request.

It has been pointed out that although Boolean functions do, in fact, accomplish the stated effects, from the systems optimization standpoint, they do not do so efficiently. It is demonstrated that Boolean combinations will yield either too much irrelevant material or too little relevant material.[38]

In the attempt to find a more flexible and optional treatment of the retrieval problem using keywords, Del Ballard (formerly of RCA) has developed a new approach to search logic. The technique is one that is easily implemented on a general-purpose computer and provides the following:

1. Retrieval of "best available" documents
2. Ranking of documents by the degree to which they match a request in terms of number of keywords
3. Upper and lower bounds on either the number of keywords or number of documents
4. Sensitivity to various attributes of keywords, e.g., rank, order, numerical magnitude, ranges of values
5. Arrangements of search terms into groups that fit the requester's thought process, including nesting of groups

It is stressed that the grammatical problem is circumvented by grouping concepts without specification of the relationships between them. It has proven to be a pliable, rough, and ready tool suitable for the general practitioner in information retrieval.

MANUAL — DEDICATION OF SPACE

A systematically arranged index can be searched by proceeding according to the rule of systematization as in the standard card or book catalog. The rule may be simple, as in alphabetic filing, or it may rely on a certain amount of prior knowledge on the part of the searcher as in chemical and biological hierarchies. In more recent developments, such as the permuted index and manual coordinate indexing cards, the searcher is guided more rapidly to a specific location of information.

A technique that is both efficient and adaptable for eventual machine searching makes use of the Dedication of Space principle. In essence, a particular geometric point or region is used to represent an accession number. The matching of like points for several terms yields documents containing those terms. This is a form of the coordination principle. The best known example is the peek-a-boo card, one version of which is a standard punched card of the IBM type coded so that each space represents a document number. Matching is achieved by aligning several term cards and noting the document numbers (holes) through which light can be seen. If no documents are found in a search, terms can be eliminated by trial and error until a "see-through" occurs. In computer searches, it is possible to generate a "minimum search criterion" which, for a given set of terms, will yield those documents containing the largest number of terms in the set.

STORAGE MEDIA

Paper Media

Paper holds a certain record for longevity as the prime medium for recording information. That it can be fairly permanent has been demonstrated by the Dead Sea scrolls. It will continue to be prominent for some time, although in recent decades it has taken unprecedented forms, such as punched cards and paper tape.

An interesting speculation presents itself. Although digital information can be stored on paper, the technique of sensitization does not make use of the intrinsic properties of the medium, i.e., holes are punched or magnetic materials are deposited. Researchers in the Soviet Union have uncovered a slight piezoelectric sensitivity in wood which may indicate possibilities for the use of paper.

In general, cards and paper tape are used either as storage media for small manual or semiautomated systems or as the input medium for large automated systems. Punched cards and punched paper tape executed by automatic writing equipment are used for the latter. Small systems use peek-a-boo cards, keysort cards, edge-notched cards, uniterm cards, or modifications of these.

Magnetic Media

Tapes

Magnetic tape is extremely well suited to the long-time storage of large quantities of data in machine-language form. It is also one of the most rapid search media. For this reason, it is used most extensively as the peripheral memory of computers and, therefore, for the storage of older, less frequently needed information. Its nature is generally more permanent and less flexible than other magnetic media.

Because it is most adaptable to digital data, magnetic tape usually works best for the storage of index terms and document numbers. There are times when it is advantageous, ac-

cording to some people, to record entire text in machine language on tape; but this is not conceivably practical except for data. However, there is a system called VIDEOFILE (an RCA development) which stores documentary and pertinent bibliographic information using video techniques. The document image is converted to a video signal and is stored with an index number. It has the advantage of furnishing rapid and direct hard-copy printout and is amenable to communication media.

Drums

Magnetic drums are better suited than tapes for the internal memory of a computer because of their more rapid (many orders of magnitude) random access. Being a lookup device, the drum is well adapted to storage of the internal functions of a computer, such as dictionaries, thesauri, and program instructions. Moreover, it is the cheapest type of internal, or main, memory.

Discs

Another dimension is added by the use of disc memory units, which have characteristics similar to drums but with a considerable gain in access time and storage space. Given an address, the information is located on a certain disc in the stack, a certain sector of the disc, and a certain band within the sector.

Magnetic Cores

Magnetic core arrays provide still faster retrieval, but the expense per unit storage space is the highest of all internal memories. Cores work well in conjunction with magnetic tape as the peripheral storage media.

Cards

Magnetic cards need to be searched serially, but they have the advantage of flexibility over tape with respect to additions or deletions. However, the search rates are slower. The cards are usually stored in cartridges and may be used in conjunction with film chips, in which case the identifying codes are in magnetic digital form whereas the information is in the form of a photographic image. Magnetic coding is sometimes used in conjunction with cards containing the printed information.

Optical Media

Microfilm Reels

Most of the forms of magnetic media have their photographic analogs and the same comparative advantages. Photographic media have the main virtue of storing information in a form ready for immediate availability (document images). This is particularly suitable for drawings, abstracts, and texts which need not be read by machines but are accompanied by machine-readable codes.

Like magnetic tape, continuous microfilm must be scanned serially and is not easily updated or otherwise altered except by the cumbersome process of editing and splicing.

In one way, film is less flexible even than magnetic tape because it lacks the erasing feature. However, for storage of nondigital materials to be displayed and read by people (as opposed to processable data), this slight difference is more than offset by the possible elimination of the middle step of the sequence: (request) → (index storage) → (document storage). The last two steps can be combined when film is used.

Microfilm Strips

A natural way to introduce flexibility into microfilm storage is to cut the film into short strips. This approach is used in the nonconventional application of Minimatrex, a system for the projection of a peek-a-boo type coordinate index onto a screen. The system is discussed under Indexing, Presentation of Indexes.

Microfilm Chips

The usual technique for breaking up microfilm is to store separate frames containing, as a rule, from one to four reduced document-image pages. The chips are generally stored in cartridges. The fully automatic handling of such devices is expensive, but a compromise is often effected by providing efficient manual retrieval of the cartridges and automatic retrieval of the chips.

Aperture Cards

More than satisfactory results have been obtained in some cases by combining film with other types of media. For example, film chips

containing document images can be inserted in magnetically coded cards (Magnacard system), providing a compact and flexible file-storage complex.

Another highly efficient combination is film and paper, i.e., punched cards of the IBM type with microfilm inserts. The cards are comparatively inexpensive; search is relatively swift; the file is flexible (easily updated and purged); and it affords all the convenience of having the index, document store, and retrieval system in one location.

Specialty Devices

IBM is developing a photoscopic disc capable of storing up to a billion bits. The speed of access is about the same as that of magnetic drums, but the large capacity makes it applicable for the kind of large storage lookup required in the processing of natural languages and machine translation. There is an erasure problem, which is minimized by coupling with a smaller auxiliary magnetic drum.

MIT is developing the Photomemory, a complex of 4- by 5-inch photographic plates, each of which can store 5 million bits of information. Reports on access time are inconsistent; generally, it appears to be a medium access time device. Access is sequential, density is high, and readout is rapid. The photographic plate is scanned by a rotating mirror which projects the image past a bank of photosensitive cells.

An electron optical device based on an effect called thermoplastic recording is under development. Still in the laboratory stage, it makes use of the properties of electron beams which have low deflection inertia, high energy, and—most important—high resolution. Information is stored as a pattern of electric charges, the field of which deforms the thermoplastic surface of a film upon subsequent heating. This means that signals can be stored with a resolution much less than the wavelength of visible light. To cope with the amplitude-variation rather than density-variation form of the recorded information (surface ripples), an optical schlieren system is used to read the image. The potential impact of this very advanced device on ISR is not yet evident.

Other devices, intended for use primarily in associative, or content-addressed, memories are neuristors, cryotons, and fiber-optical devices. Associative memories tend to be slower in arithmetic operation than in conventional storage units. However, because of the capability of direct access to content rather than via a location code, the potential for nonnumerical information processing is promising.

DISSEMINATION

One avenue of communication between people and information is retrieval, i.e., letting scientists come to the information and helping them locate what they want. Another way of augmenting this communication is to seek ways in which important information can be channeled to people who might need it. The emphasis here is on "might" for there is no way of knowing with assurance who will benefit by what.

Most of the details of automatic Selective Dissemination of Information (SDI) have been supplied by IBM.[17] While it is agreed with IBM that selective dissemination must be an essential part of any technological organization, there is no assurance that it will promote nearly perfect matching between need and answer to the need.

DIGRESSION ON SCIENTIFIC DISCOVERY

Richmond, writing on the relation between information retrieval and scientific method and issuing a plea for more recognition of the latter,[32] stresses the role of fortuity in scientific discovery. She sites several historical examples of scientific events and, generalizing, describes two such events in which the cards do not fall to the researcher's advantage: near discovery, where a scientist hovers for years on the threshold of a breakthrough, working all around it but failing to unlock the solution; premature discovery, where a scientist discloses a major phenomenon either without realizing it or in the face of an over-all state of the art that is not ready to accept his discovery. One reason for the latter is that what may later prove to be an immensely important application is not at all apparent at the time. Some eminent discoveries have at the time of conception been dismissed as interesting novelty or at best "vital to our understanding of the universe about us but of no conceivable utility."

Sometimes rediscovery in a sympathetic environment carries more weight than the original discovery, which is afterwards forgotten except by the probing researcher who might have unearthed it from the musty archives of "interesting novelties."

What is involved, of course, are those most complex and very human patterns of thought, deduction and induction. Richmond expresses doubt that it will ever be possible to " . . . convey deductive and inductive reasoning in sufficiently abbreviated form . . . ," or to ". . . perform retrieval of the kind the historian performs . . . " (i.e., identify a premature discovery as a discovery in light of subsequent advances). One is confronted again with the lead that capabilities, in terms of equipment, have over human understanding.

Present information retrieval and dissemination do not do the scientist's thinking for him (and this is Richmond's prime thesis); they give him written accounts of work somewhat similar to his own. Retrieval and dissemination will yield the expected far more often than the unusual, the reason being that the systems cannot draw meaningful relationships between any two heretofore dissociated things.

Therefore, assuming a retrieval or dissemination system operates on key words, if the words "aardwolf" and "zymosis" have never been related, the system will not coordinate them (except to supply Webster's Dictionary or the Encyclopaedia Britannica).

The system may provide separate treatises on each, but these will serve only to enhance the requester's separate expertness on both items. As an instructive exercise, the system might be programmed for random coordination, in which case, assuming a typical 5000-keyword system, there is one chance in 12,497,500 that the words will be associated. Now a slight connection is seen between what machines can do and the human trial and error process and how infinitely more educated the latter.

There is a need for further study into the ways in which scientists benefit by the work that has gone before. To accelerate and stimulate the scientist's learning process, this should be the task of a dissemination system. A concrete approach would be the re-creation by simulation of some of the significant historical breakthroughs and discoveries as well as those that did not happen because the information was not available at the right time. The objective would be to utilize the imposing edge held in retrospect today over the researcher in his day. The task would involve computer simulation of the historical event, drawing on both antecedent and postcedent (via feedback) literature and events. It is hoped that the outcome would include (1) a better understanding of the creative thought process and what could be done to aid it, (2) a manifold improvement in efficiency of the simulated over the real process of discovery, e.g., re-creation of an order of magnitude sooner than the original occurrence. If successful, the profit would be a more efficient homing-in on future discovery via the literature route.

The question of meeting user need is taken up again briefly under Storage and Retrieval.

DESCRIPTION OF SELECTIVE DISSEMINATION OF INFORMATION (SDI)

There are times when a scientist's best dissemination system is a good friend. Casual conversation has often produced the vital key in a field so remotely diverse as to be otherwise overlooked. Such might have been the case when the problem of moving film with high precision at high acceleration was solved by digital positioning techniques developed for an automatic drilling machine.[29]

The problem of a formal dissemination system is to be as good a friend in a continuous-operation and systematic mode. A formal dissemination system should always be aware of the scientist or engineer's activity, keeping pace with it and alerting him to anything by which he may profit yet without wasting his time with documents of no interest. This is a formidable assignment.

The usual approach of a formal dissemination system is to screen potentially relevant materials through the standard gradations of presentation, i.e., key word, abstract, and document. The key word operation is handled before the relevant material reaches the scientist by means of a so-called interest profile. The interest profile concept is a useful one for match-

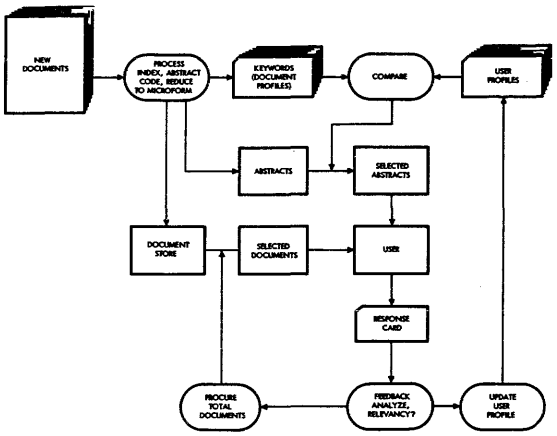


Figure 3. Flowchart of Selective Dissemination.

ing operations in general. The profile is essentially a list of key words or data used to describe objects to be compared, e.g., a person, his attributes, his information needs, his abilities, a document, or needs of an organization.

Figure 3 illustrates the workings of a selective dissemination system used to match information of incoming documents with user needs.

FEEDBACK

Document profiles remain fairly stable, while recipient profiles are subject to continual modification through feedback. In the SDI system, response cards are provided along with abstracts of documents whose profiles suitably overlap the recipient profile. The recipient returns the response card with an indication that he (1) had no interest in the document, (2) found the abstract useful and sufficient, (3) found the abstract interesting and requests the total document. His profile is then updated on the basis of his responses.

The depth of matching, i.e., the extent to which the recipient profile and document profile overlap, is a controllable parameter.

EXPANSION OF SDI

The skill profile now employed by Space and Information Systems Division is one example of the profile matching concept. The skill file can be searched when management wants to know who is knowledgeable in a given area. The system furnishes the name and location of the person whose particular talents most nearly correlate with those sought by the organization. The same principle can be applied to new projects, matching the contemplated project profile with the profiles of projects already underway.

ABSTRACTING

Webster's definition, modified to fit documents, in a certain sense exemplified the avowed purpose of the abstract: a text (as concise as possible) that comprises or concentrates in itself the essential qualities of a document or of several documents (generalizing the latter to a book of several chapters, state-of-the-art survey, blanket review, etc.).

A little investigation reveals that within the field of documentation there is not a clear operational definition of abstracting to complement the conceptual definition above, and indeed several researchers are occupied with finding one. Some other things that need to be considered are the purpose and advisability of abstracting, standardization, and the desirability of standardizing abstracts. These factors must be understood before delving too deeply into the ultimate aspect of abstracting mechanization.

PURPOSE OF ABSTRACTING — ABSTRACTING VERSUS INDEXING

For the purpose of definition, it is important to differentiate functionally between an abstract and an index, a detailed descriptor index in particular. One can almost visualize a document being screened through several gradations of processing. At one end would be the single subject heading or "catchword" (Schlagwort). At the other end of the range is the total document itself. In order of increasing depth of representation, the intermediate steps would be a bibliographic index, a complete document profile as used in selective dissemination, a more detailed informative index using descriptors or Uniterms, scope notes, a detailed index with roles and links, a so-called supplied title (a title written by the document processor), a one- or two-sentence synopsis, a telegraphic abstract, and a detailed summary or abstract. However, the user will not want to be bothered with more than three of these includ-

ing the total document. Under the proper circumstances, the continuum seems to fall spontaneously into the parts necessary for that set of circumstances. For example, a trade journal, well attuned to the needs of its readers, may give a list of one-sentence descriptions of worthwhile reading matter (supplied titles) providing the reader with sufficient stimulus to procure the document. However, if a library is to get information to a large and diverse clientele, it cannot hope to cater to the individual user as the editors of a trade journal can to their readers. It cannot expect to handle an influx of up to a thousand documents per day, evaluate their content, and dramatize their highlights in the way that a trade journal can. Not knowing the specific needs of the user, the alternative extreme would be to give him every potentially pertinent document. Such a system would entail a disproportionate waste of time.

The index and search techniques will help the user to find things which might be what he thinks he wants. The dissemination operation will help him receive things which might be what he had not thought of or did not know about. To conserve the user's time, abstracts of those documents with possible or latent interest are supplied to him. Therefore, an abstract is not an index, for an index indicates what a document is about, or it tells where the information is located. An abstract is not a document in the strict sense of the word for it does not tabulate data nor does it give detailed hookup and assembly instructions for an instrumentation complex. An abstract is not identical with either an index or a document, but it incorporates features of both.

STANDARDIZATION OF ABSTRACTING

Several studies are being conducted to find operational definitions of abstracting. The two mainstreams of effort are directed toward examining the effect of various abstracting philosophies on requesters and their satisfaction and toward evaluation of different abstracts of the same or similar documents. Several parameters are involved, and in most studies one or more of these are controlled. In one investigation,[8] the size of the natural language abstracts is held to a fixed percentage (10 per cent) and the work of different abstracters is compared.

A unique part of the same investigation is devoted to the study of "term diagrams," which incorporate the "road map" idea of Loren Doyle (see Indexing) to show quantitative and conceptual associations between key words. These studies are aimed at eventual automation of document condensation.

The American Institute for Research is conducting an extensive inquiry into the operational definition of abstracting, the development of rules guided by task performance on the part of both abstracter and user, and the evaluation of efficiency.[8] A survey is being made among abstracting services, publishers, and users to gather ideas from those experienced in the field. Simulated job conditions are set up to measure user comprehension and proficiency under these conditions, given both full and abstracted texts.

AUTOMATIC ABSTRACTING

The principal efforts in automatic abstracting are being carried out at Thompson-Ramo Wooldridge, Inc. System Development Corp., Planning Research Corp., and at International Business Machines. Without going into the detailed procedures for abstracting, the primitive approach is to first find statistically significant keywords (content words with a high frequency of occurrence) and then find those sentences which contain the greatest number of such keywords. The results so far are conspicuous by their lack of coherence; partly because of habit patterns in reading, they seem to present disconnected ideas with a loss of flow between the ideas. Of course, the abstracts suggest rather than condense or represent the document contents. Nevertheless, they are surprisingly good on occasion. The problems just mentioned have been instrumental in the inception of more probing studies into the nature of human abstracting.

CRITICAL COMMENT ON AUTOMATIC ABSTRACTING

As an incidental preface, the importance of the abstract in general should be considered. It would be difficult to overemphasize the place of the abstract or document summary in the mechanized system of the future. Apart from its utility to the user in some form, it will be the

prime medium of the system, even the main unit of communication. An abstract is a useful and logical unit because it can be printed on a frame of microfilm or on a coded (punched or magnetic) card or on a single sheet of paper.

Even advanced mechanized systems will probably use manually produced abstracts. Automation of the abstracting process will be one of the last stages in the evolution of fully automated systems. The reason is evident after a moment's reflection. Difficult as the problem of automatic language translation is, it is one order of complexity lower than the job of abstracting. In many ways the difference between the two is analogous to the disparity between a dictionary translation and a translation turned out by a truly bilingual specialist in the field. The latter, if a good translator, will not simply convert words or even phrases. He will restate the author's thoughts in the new medium. The abstract must do this same thing.

Automatic abstracting comprises the ultimate refinement of automatic translation and automatic indexing. It must translate ideas from expanded to compressed language, and it must be selective. It must also be endowed with a product all its own, the attribute of paraphrase.

REPRODUCTION AND DISPLAY

There are a number of hard copy reproduction methods available, including:

Copy Camera	Ozalid
Ditto Process	Photoengraving
Electrography	Photogravure
Letter Press	Photolithography
Lithography	Rotogravure
Mimeograph	Thermofax
Offset Printing	

There is a method to suit almost every conceivable need. In general, the obtaining of higher-quality, cleaner copy calls for a greater investment.

The situation is not so well resolved in the case of display systems. The technology of display has generally lagged behind the advancement of data processing and transmission. However, government and industry have begun to exhibit a resurging interest in computer-generated large-scale displays. The development of high-capacity command and control systems within the Air Force has emphasized the need for sophisthicated data-presentation subsystems.[14]

There are three main areas of technological effort:

Projection, using a stable light modulator, such as film or selenium plate.

Light valve, also a projection device, with direct electronic control of the image. This has advantages of speed and elimination of expensive film and the disadvantages of resolution and brightness (two urgently requisite factors in IS&R display).

Electroluminescence, no projection, experimental prototypes only. Display surface acts both as light source and modulator.

In general, for the specific applications of IS&R, there are no suitably inexpensive systems for reproduction and display. Essentially what is needed is a low-cost, low-volume device to meet the demands of a circulating library replaced by a noncirculating, automatic transmission-display-reproduction system. Admittedly, for many applications, microfilm display can be made inexpensive, particularly where it is sufficient to exhibit an entire sequence of frames or film. But for IS&R, which is a selective application, research and production have not come up with adequate equipment.

COMMUNICATION LINKS

Hardware development in IS&R will ultimately make it unnecessary that a requester be physically present at the information center. Links for communication between remote request stations and the center, which are now available or in advanced stages of development, range from conventional telephone lines to systems employing microwave devices. A closed-circuit television system (CCTV) will lend itself very readily to application in IS&R. Therefore, the communication system might be as simple as one permitting a requester to initiate a search by telephone and whose results, in the form of abstracts, are returned to him by telephone or mail. With abstracts and documents stored on microfilm and a CCTV/telephone system, the requester could, in a more advanced system, view abstracts or the document itself.

The ultimate in a communication link between the user and the information center would permit the user to deal directly with an automatic IS&R unit (computer) to initiate a search, call up on a viewing screen the documents resulting from the search, and reproduce these in the form of hard copy. While all the functions involved are conceptually simple, considerable development and system integration effort must be accomplished before the ultimate system becomes a reality, especially in the development of CRT line scanning (800 to 1200 lines).

In the area of volume hard-copy reproduction, to take one example, the state-of-the-art survey for this report revealed no systems that could be called ideally suited to IS&R if one considered high-cost trade-offs for systems such as electrostatic-video printing. Among the lesser problems, considerable work in human factors for such a system might be done. Detailed answers to questions of viewing-screen brightness and contrast and ambient illumination should be made available. While all the individual functions involved are conceptually simple, considerable development and system-integration effort must be accomplished before this becomes a reality. Much groundwork has been done,[32] but the extent to which experience in viewing conventional TV visual presentations can be applied to the prolonged reading of text for an ideal IS&R visual link is questionable.

SUMMARY

The status of information storage and retrieval is summarized in Figures 4 (Equipment) and 5 (Techniques). The sizes of the boxes are drawn to indicate their relative "strength" of development of the given area. Note that the "Equipment" circle is drawn larger than the "Document Processing" circle to show the greater strength of development in this area. A notable exception is the equipment for graphics and display, microfilm reading, and reproduction in particular.

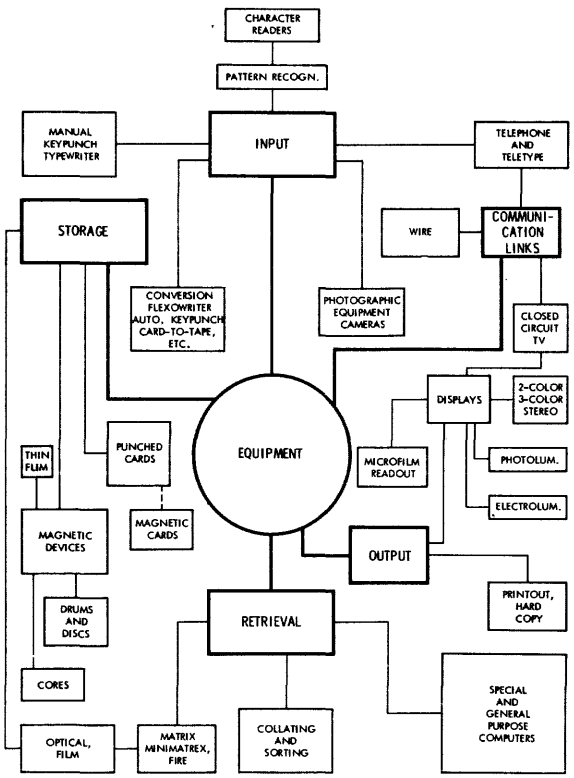


Figure 4. Equipment State of the Art.

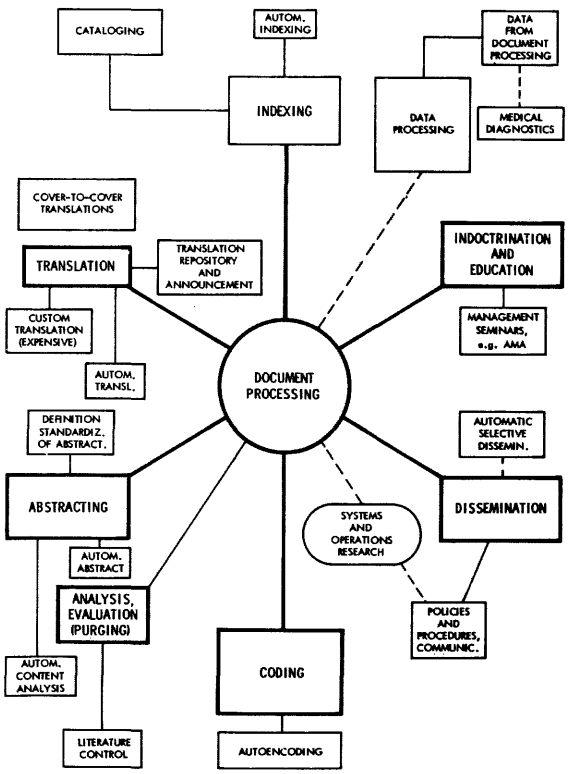


Figure 5. Technique State of the Art.

In two respects, IS&R is not yet an integrated science in the sense of chemistry or even such an interdiscipline as bionics. First of all, the contributing disciplines are diverse and overlapping, but hardly unified. Secondly, although many nonconventional systems are in operation throughout the world, they have little in common by way of a unifying description.

The most glaring deficits are the software lag and lack of a truly empirical basis. Today is part of an era of machines that can analyze data, read and recognize printed symbols, recognize audible and verbal signals, be conditioned and learn, teach humans or other machines, and even make decisions. Thin film memories are reducing access times to the nanosecond realm. A billion items of distinct information can be stored in a space the size of a paperback novel. But trailing far behind these amazing creations in the equipment field are the concepts of how people communicate with each other, particularly when the scientist or engineer presenting his results or theory on paper does not know who in what remote corner will want to read it, or when the person seeking information has some vague unfulfilled desire and does not know who in what remote corner has the happy cure for his troubles. Each one hopefully relies on those who are entrusted with getting the answer to the need and finding the market for the goods.

In the meantime, advanced thinkers are passing over the concepts-equipment gap and are developing theories for conceptual machines not yet in existence. And—emphatically—they are needed. But concurrent with these efforts is a greater need for less speculation and conjecture, less projection of pet ideas and systems, and more experimental investigation and scientific method.

The ultimate criterion of how well a retrieval system works is the extent to which retrieved information meets the need of the user. From the outset one is confronted with problems because user need is already an idealization. The corresponding measure of how well this need is met might be called user satisfaction, which is also an idealization. Because user need and user satisfaction are highly unquantitative and highly indeterminate, they must be approximated. The expression of user need is a request and the vague notion of user satisfaction is simulated by the notion of relevancy. The situation might be visualized as in Figure 6.

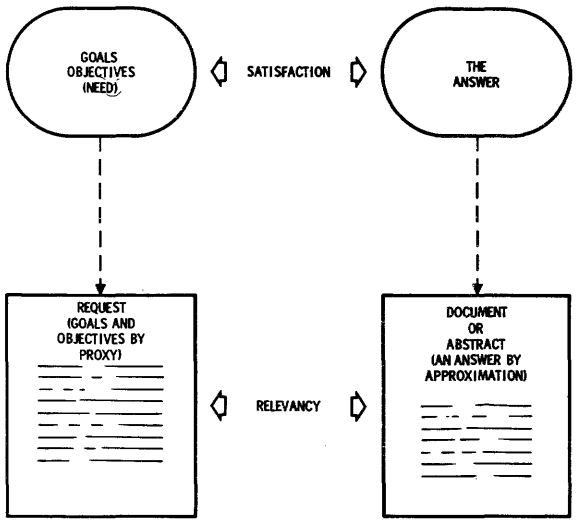


Figure 6. Contrast Between Ideal and Real Information Retrieval.

BIBLIOGRAPHY

1. ARNOVICK, G. N., "Automatic Analysis and Synthesis of Specifications for Machine Systems in Information Storage and Retrieval," presented at the Annual Meeting of the American Documentation Institute, Miami, Florida, December 10–15, 1962.
2. ARTANDI, S., and T. C. HINES, "Roles and Links—Or Forward to Cutter," American Documentation 14(1): 74–77, January 1963.
3. BAGG, T. C., and MARY E. STEVENS, Information Selection Systems Retrieving Replica Copies: A State-of-the-Art Report, National Bureau of Standards Technical Note 157, December 31, 1961.
4. BOBROW, D. G., "Syntactic Analysis of English by Computer—A Survey," Proceedings of the Fall Joint Computer Conference, November, 1963, pp. 365–388.
5. BORKO, H., and MYRNA BERNICK, "Automatic Document Classification," Journal of the Association for Computing Machinery 10(2): 151–162, April 1963.

6. BORKO, H., "Some Recent Experiments in Automatic Document Classification," Linguistic Research Seminar, The RAND Corporation, Santa Monica, California, January 9, 1964; "Research in Automatic Generation of Classification Systems" (1964 SJCC).
7. CORNERETTO, A., "Associative Memories; A Many-Pronged Design Effort," Electronic Design ( ) : 40–55, February 1963. (An excellent introduction and survey.)
8. Current Research and Development in Scientific Documentation, No. 11, National Science Foundation, Office of Science Information Service, Washington, D. C., November 1962.
9. DOYLE, L. B., "Semantic Road Maps for Literature Searchers," Journal of the Association for Computing Machinery 8(4) : 553–578, October 1961.
10. EDMUNDSON, H. P., and R. E. WYLLYS, "Automatic Abstracting and Indexing—Survey and Recommendations," Communications of the Association for Computing Machinery 4(5) : 226–234, May 1961.
11. FORD, J. D., JR., and E. H. HOLMES, A Comparison of Human Performance Under Natural Language and Term Diagram Procedures for the Production of Report Summaries, TM–622, System Development Corporation, February 6, 1962.
12. GARFIELD, E., "Citation Indexes for Science," Science 122: 108, 1955.
13. GARFIELD, E., "Citation Indexes in Sociological and Historical Research," American Documentation 14(4) : 289–291, October 1963.
14. Guide for Group Display Chains for the 1962–1965 Time Period (advance copy), Directorate of Engineering, Rome Air Development Center, New York, March 27, 1962.
15. HAYS, D. G., Grouping and Dependency Theories, RM–2646, The RAND Corporation, Santa Monica, California, September 8, 1960.
16. HERNER, S., and M. HERNER, "An Experiment in the Use of Reference Questions in the Design of a Classification System," report to the National Science Foundation, Washington, D. C., Herner & Co., January 1962.
17. IBM General Information Manual: Selective Dissemination of Information, International Business Machines Corporation, White Plains, New York, 1962.
18. IBM Workshop on Information Storage and Retrieval, San Jose, California, June 26–29, 1962.
19. KENT, A., "Machine Literature Searching and Translation. An Analytical Review," Advances in Documentation and Library Science, Vol. III: Information Retrieval and Machine Translation, Inerscience, New York, 1960, pp. 13–236.
20. KESSLER, M. M., "Bibliographic Coupling Between Scientific Papers," American Documentation 14(1) : 10–25, January 1963.
21. KLEIN, S., and R. F. SIMMONS, "Syntactic Dependence and the Computer Generation of Coherent Discourse," Mechanical Translation 7(2) : 50–61, August 1963.
22. LUHN, H. P., Auto-Encoding of Documents for Information Retrieval Systems, International Business Machines Corporation, Yorktown Heights, New York, 1958.
23. LUHN, H. P., The Automatic Derivation of Information Retrieval Encodements from Machine-Readable Texts, International Business Machines Corporation, Yorktown Heights, New York, 1959.
24. MARON, M. E., and J. L. KUHNS, "On Relevance, Probabilistic Indexing and Information Retrieval," Journal of the Association for Computing Machinery 7(3) : 216–244, March 1960.
25. MARON, M. E., "Automatic Indexing—An Experimental Inquiry," Journal of the Association for Computing Machinery 8(7) : 404–417, July 1961.
26. MORSE, R. D., "Information Retrieval," Chemical Engineering Progress 57(5) : 55–58, 1961. (This is the best account of links and role indicators.)
27. Pacific Southwest Navy Research and Development Clinic, Panel on Information Systems and Computer Technology, Berkeley, California, October 16–18, 1963.

28. RCA–NBC, Before the Federal Communications Commission, Petition of Radio Corporation of America and National Broadcasting Company, Inc., for Approval of Color Standards for the RCS Color Television System, June 25, 1963.
29. RENOLD, W., Journal of the Society of Photographic Instrumentation Engineers 1(2): 44–48, December 1962/January 1963.
30. RESNICK, A., "Relative Effectiveness of Document Titles and Abstracts for Determining Relevance of Documents," Science 134(3484): 1004–1005, October 6, 1961.
31. RESNICK, A., "Comparative Effect of Different Educational Levels on Indexing in a Selective Dissemination System," International Business Machines Corporation (in press).
32. RICHMOND, PHYLLIS A., "What Are We Looking For?" Science 139: 737–739, February 22, 1963.
33. SALTON, G., "Some Experiments in the Generation of Word and Document Associations," Proceedings of the Fall Joint Computer Conference, December 1962.
34. STERN, J., "An Application of the Peek-A-Boo Principle to Information Retrieval," Proceedings of the Symposium on Materials Information Retrieval, Dayton, Ohio, November 28–29, 1962.
35. STILES, H. E., "Progress in the Use of the Association Factor in Information Retrieval," Proceedings of the Symposium on Materials Information Retrieval, Dayton, Ohio, November 28–29, 1962.
36. SWANSON, D. R., "Information Retrieval: State-of-the-Art," Proceedings of the Western Joint Computer Conference, Los Angeles, California, September 9–11, 1961.
37. SWANSON, D. R., "State-of-the-Art Paper," Conference on Libraries and Automation, sponsored by the National Science Foundation, The Library of Congress, The Council on Library Resources, Inc., Airlie Foundation, Warren, Virginia, May 26–29, 1963.
38. VERHOEFF, J., W. GOFFMAN, and J. BELZER, "Inefficiency of the Use of Boolean Functions for Information Retrieval Systems," Communications of the Association for Computing Machinery 4(12): 557–558; 594, December 1961.
39. WILLIAMS, J. H., JR., "A Discriminant Method for Automatically Classifying Documents," Proceedings of the Spring Joint Computer Conference, November 1963, pp. 161–166.

TRAINING A COMPUTER TO ASSIGN DESCRIPTORS TO DOCUMENTS: EXPERIMENTS IN AUTOMATIC INDEXING

M. E. Stevens and G. H. Urban
National Bureau of Standards
Washington, D. C.

INTRODUCTION

During the past five to ten years, increasing interest has developed in the use of machines as a substitution for human intellectual effort in the indexing or classification of the subject content of documents. In terms of practical applications, the greatest interest to date has been in keyword indexing from the significant words actually occurring in titles, abstracts, or full texts. Pioneering use of computers for this purpose, by Luhn[8] and Baxendale,[2] has been followed by the development of a number of KWIC (keyword-in-context) and similar programs.

This is "derived" or "derivative" indexing as opposed to "assignment" indexing.[6,8] In derivative indexing, the indexing terms used are identical with words chosen by the author himself. In assignment indexing, the terms used are typically drawn from a classification schedule, a subject heading list, or a thesaurus of descriptors, and they are assigned on the basis of decision-judgments about the appropriateness of these terms as indicative of the subject matter of the document in question. Approaches to indexing by computer based upon assignment of indexing terms or upon automatic classification techniques have been reported by Baker,[1] Borko,[3,4] Maron,[9,10] Needham,[11] and Swanson,[15] among others.

Small-scale experiments in automatic assignment indexing have also been conducted at the National Bureau of Standards in recent months. The SADSACT (*S*elf-*A*ssigned *D*escriptors from *S*elf *A*nd *C*ited *T*itles) method is based on the two working hypotheses:

1. That, given only a "teaching sample" of previous human indexing of representative items, the machine can derive statistics of association between substantive words used in the titles or abstracts and the descriptors assigned to these items such that patterns of substantive words in the titles and cited titles of new items can be used for automatic assignment of descriptors to these new items, and
2. That, as in the case of citation indexes, the bibliographic references cited by an author can provide useful clues to the subject of his own paper.

In addition, we hope to demonstrate an automatic indexing technique that will minimize time and cost of preparation by limiting the amount of input material required and by capitalizing on byproduct data generation from one-time keystroking of descriptive cataloging information plus the text of the titles of bibliographic references cited in new items.

THE TRAINING SEQUENCE AND INDEXING PROCEDURE

In training the computer to assign descriptors to our documents, we first select a sample, presumably representative, of items already

in our collection. These have all previously been indexed to some pre-established indexing-term vocabulary, as the result of human subject content analysis. In our experiments to date, the "teaching sample" has consisted of approximately 100 items drawn from a collection of about 10,000 technical reports in the fields of information retrieval and potentially related research including computer design, programming languages, mathematical logic, linguistics, pattern recognition, psychology, operations research, and the like. These teaching sample items are ones that had all previously been indexed by analysts at ASTIA (now the Defense Documentation Center) in the year 1960. Descriptive cataloging information, abstracts, and lists of descriptors assigned were available. This information was keystroked on Flexowriters and the texts of the titles, descriptors, and abstracts were converted to punched cards for input to the computer.

In programming the SADSACT method for trial on the IBM 7090–7094 computers, the cards for each of the documents in the teaching sample are read into the computer, one item at a time. A list of descriptors assigned to these items is formed, and the frequency of use of each descriptor for the sample as a whole is tallied. The text of the title and abstract of each document is then processed against a "stop" list to eliminate the common and non-significant words. Each remaining word from document title and abstract is associated with each of the descriptors previously assigned to that document. Sorted lists of English words and their descriptor co-occurrence relations are formed and read out, using three output tapes.

When all the documents in a teaching sample have been processed, the next step is to identify the "validated descriptors"—that is, those that have occurred three or more times in a 100-item sample.* (The "names" of other descriptors assigned to not more than one or two items are retained as "candidate descriptors" but word co-occurrence data is not retained for these.†) For these validated descriptors, the word-descriptor association lists are then merged into a master vocabulary list which gives for each word the identity of the descriptors with which it co-occurred and the relative frequency of its co-occurrence with each descriptor.

The SADSACT automatic indexing method, therefore, uses an *ad hoc* statistical association technique in which each word may be associated either appropriately or inappropriately with a number of different descriptors. In the subsequent indexing procedure, reliance is placed on patterns of word co-occurrences and on redundancy in new items to depress the effects of non-significant or inappropriate word-to-descriptor associations and to enhance the significant ones.

The SADSACT indexing procedure is carried out as follows. The text of the title of a new item and of titles cited as bibliographic references by the author is keystroked, and the by-product punched paper tape is converted to cards for input to the computer. This input material is processed against the master vocabulary list to yield, for each word matching a word in the vocabulary, a "descriptor-selection-score" value for each of the descriptors previously associated with that word. When all words from titles and cited titles have been processed, the descriptor scores are summed and for some appropriate cutting level,‡ those descriptors having the highest scores are assigned to the new item.

The actual score value derived reflects both a normalizing factor (based, for example, on the ratio of the number of previous co-occurrences of this word with a particular descriptor to the

* In our first teaching sample, 219 different descriptors had been used by the DDC indexers, and 72 occurred for three or more items in the sample. Our second teaching sample, formed by substituting different items for 30 items withdrawn from the first to be used as test material, gave 221 descriptors, of which 70 were validated.

†One of the distinctions to be emphasized in assignment as opposed to derivative indexing is that although a word or phrase occurring in a title or in text may coincide orthographically with the "name" of a descriptor or subject heading, its occurrence may or may *not* result in the assignment of that descriptor to the item in which it occurred.

‡In machine tests to date, the cutting level has been arbitrarily set so that exactly twelve descriptors are assigned to each item. However, since these are printed out in decreasing order of selection-score values, results of other cutting levels can readily be computed.

number of different words co-occurring with that descriptor) and a weighting formula that gives greater emphasis to words occurring in "self"-title (the author's own choice of terminology) than to those occurring in cited titles. Similarly, greater emphasis is given to words that coincide with the names of descriptors. The latter weighting enables the assignment of "candidate" as well as "validated" descriptors, if words so coinciding occur several times in the input material. An extension of this feature would allow similar treatment to be given to n-tuples of substantive words (such as "automatic character recognition") that had occurred with some frequency in the teaching sample material, so that "potential descriptors" could be used to provide change and growth in the indexing vocabulary.

RESULTS TO DATE

In a previous informal report, we have noted the results of applying the SADSACT method to eight new (1963) items to our first 100-item teaching sample, consisting of selected documents indexed by DDC in the spring of 1960. These results were disappointing in terms of comparison against discriptors assigned by DDC indexers, largely because of changes in indexing philosophy and vocabulary such that many descriptors used by human indexers in 1963 were not available to the machine. However, 49% of the descriptors judged by a subject matter specialist to be relevant to these items were correctly assigned by the computer program.

Our first report also covered tests on 30 "new" items drawn from the teaching sample itself in order to provide, quickly, a basis for evaluation in which the indexing vocabulary could be held constant. Where titles and abstracts had been used for the teaching sample procedure, for this test the titles and cited titles only were used, so that a reasonable proportion of the input was new rather than

TABLE 1. Machine "hits" against DDC assignments, tests run against T.S. #1 only

Short Title of Item	% Hits All DDC Descriptors	% Hits Descriptors Available to Machine	% Hits Validated Descriptors
Perceptrons and the theory of brain mechanism	37.5	42.9	60.0
(Pattern recognition with an adaptive network	33.3	33.3	50.0)
Pattern recognition with an adaptive network	50.0	50.0	50.0
Invariant input for a pattern recognition machine	25.0	33.3	33.3
An adaptive character reader	37.5	60.0	75.0
Theory of files	25.0	50.0	50.0
Learning in pattern recognition	33.3	50.0	50.0
Reorganization by adaptive automation	50.0	66.7	66.7
Finite state languages formal representations	20.0	20.0	20.0
Some computer experiments with predictive coding	45.5	55.6	71.4
On categorical and phrase structure grammars	50.0	66.7	66.7
Error correcting codes from linear sequential	33.3	42.9	75.0
Method of improving organization in large files	57.1	66.7	100.0
Machine program for theorem proving	60.0	75.0	100.0
Pattern target analysis	0.0	0.0	0.0
Tables of Q functions for two Perceptron models	25.0	33.3	40.0
Lineal inclination in encoding information	22.2	25.0	40.0
TOTAL	549.2	713.1	858.1
AVERAGE	34.3	44.6	53.6

TABLE 2. Machine "hits" against DDC assignments, tests run against both T.S. #1 and T.S. #2

	T.S. #1			T.S. #2		
	% All	% Available	% Validated	% All	% Available	% Validated
Flowgraphs for nonlinear systems	14.3	20.0	33.3	14.3	20.0	33.3
Activity in nets of neuron-like elements	9.1	16.7	33.3	18.2	33.3	66.7
Survey of cueing methods in education	42.9	50.0	60.0	42.9	50.0	60.0
Facet design and analysis of data	37.5	42.9	50.0	12.5	14.3	16.7
Automatic recognition of speech	25.0	28.6	66.7	25.0	28.6	66.7
Problems in nonlinear control systems	33.3	50.0	50.0	16.7	25.0	25.0
Guiding principles for time and cost	40.0	100.0	100.0	40.0	100.0	100.0
Approach to pattern recognition using linear threshold devices	44.4	50.0	57.1	33.3	37.5	42.9
Simultaneous multiprogramming of electronic computers	28.6	40.0	66.7	21.4	30.0	50.0
Design and study of correlation instrumentation	22.2	44.4	80.0	22.2	44.4	80.0
The learning model approach	27.3	30.0	33.3	27.3	30.0	33.3
Soviet computer technology	50.0	50.0	60.0	50.0	50.0	60.0
Computer applications of Boolean functions	37.5	75.0	75.0	25.0	50.0	50.0
Sequential encoding and decoding	36.4	36.4	50.0	27.4	27.4	37.5
Design of a language for Simplex system users	75.0	75.0	75.0	75.0	75.0	75.0
Automated teaching a review	55.6	55.6	71.4	55.6	55.6	71.4
TOTAL	579.1	764.6	981.8	506.8	671.1	888.5
AVERAGE	36.2	47.8	61.4	31.7	41.9	55.6

"source" material. For these 30 items, the average "hit" accuracy was 64.8%. This included the occasional assignment of candidate descriptors in cases where words in title or cited titles coincided with the names of candidate descriptors frequently enough to build up a score above the cutting level. (For example, the candidate descriptor "Sleep" was assigned to an item entitled "Human performance as a function of the work-rest cycle.") Considering validated descriptors alone, the machine assignment accuracy was 77.4%.

These results were suspect, however, because of the probable source material bias. Accordingly, substitutes were prepared for each of these first "test" items to form a second teaching sample consisting of 70 items from teaching sample #1 and 33 additional items. Items from teaching sample #1, now genuinely "new," and some additional items were then run against teaching sample #2. Items new to both samples were also run against sample #1. Results for 101 machine tests on 59 different items are shown in Tables 1, 2, and 3 in terms

TABLE 3. Machine "hits" against DDC assignments, tests run against T.S. #2, both with cited titles and with abstracts

	Cited Titles			Abstracts		
	% All	% Available	% Validated	% All	% Available	% Validated
Construction of convolution codes by sub-optimization	66.7	66.7	66.7	66.7	66.7	66.7
Iterative circuit computers	40.0	40.0	40.0	40.0	40.0	40.0
Use of context cues in teaching machines	20.0	25.0	28.6	20.0	25.0	28.6
Model for communication with learning	42.9	42.9	50.0	28.6	28.6	33.3
Word length and intelligibility	28.6	33.3	40.0	14.3	16.7	25.0
Machine correction of garbled English text	71.4	71.4	71.4	85.7	85.7	85.7
Multiple task study on automatic data processing	50.0	50.0	50.0	75.0	75.0	75.0
Independence of attributes in memory	50.0	66.7	66.7	25.0	33.3	33.3
Use of models in experimental psychology	42.9	50.0	60.0	28.6	33.3	40.0
Linear recurrent binary error correcting codes	42.9	42.9	50.0	57.1	57.1	66.7
Adaptive sample data systems	42.9	50.0	75.0	28.6	33.3	50.0
Common programming language task	80.0	80.0	80.0	80.0	80.0	80.0
Some requirements for production control data	14.3	50.0	100.0	14.3	50.0	100.0
Remembering the present state of a number of variables	25.0	25.0	33.3	25.0	25.0	33.3
Papers on the APT language	57.1	80.0	80.0	57.1	80.0	80.0
Topics in the theory of discrete information channels	100.0	100.0	100.0	100.0	100.0	100.0
Some aspects of problem solving	28.6	28.6	33.3	28.6	28.6	33.3
Predictive model for self-organizing systems	25.0	25.0	33.3	50.0	50.0	66.7
Adaptive switching circuits	29.9	38.5	55.6	29.9	38.5	55.6
Techniques of pictorial data reduction	71.4	83.3	83.3	57.1	66.7	66.7
Combining of classes condition in learning	37.5	50.0	60.0	50.0	66.7	80.0
On some relations between human engineering	57.1	57.1	66.7	42.9	42.9	50.0
Human performance as a function of the work-rest cycle	9.1	14.3	25.0	18.2	28.6	50.0
The simulation of human thought	85.7	85.7	85.7	71.4	71.4	71.4
Studies of communication channels	14.3	16.7	50.0	14.3	16.7	50.0
Concept of the ideal observer in psychophysics	44.4	44.4	50.0	33.3	33.3	37.5
Learning in random nets	—	—	—	100.0	100.0	100.0
TOTAL	1177.7	1312.5	1544.6	1241.6	1368.1	1609.2
AVERAGE	45.3	50.5	59.4	45.9	50.7	59.6

of the percentages of descriptors originally assigned by DDC indexers which were also assigned by machine. These results show, even more than had been expected, a significant drop below the "hit" accuracy obtained when the same items were run under partially "source material" conditions.§

The overall average "hit" accuracy for these tests is only 40.1%, considering all descriptors assigned to these items by DDC. However, in spite of the use of test items drawn from the same time period as the teaching samples in order to maximize consistency of indexing vocabulary, 19.1% of the descriptors assigned by DDC were not available to the machine. When corrected for this not-available factor, the average "hit" accuracy was 48.2%. Other descriptors used by DDC for these items were available to the machine only as candidate descriptors and could only be assigned if input words for the test items coincided with the names of these descriptors. Considering only the descriptors that were freely available to machine, the validated descriptors, the computer assignment accuracy was 58.1%. In only one test case, for these 59 items, were no hits achieved by the machine method ("Pattern target analysis").

Figures 1 and 2 represent, respectively, machine assignments for title-and-cited-titles and for title-and-abstract of an item subsequently evaluated by several members of our staff, as shown by initials. Checks to the left of the descriptors indicate agreements with DDC assignments. The machine missed three DDC assigned descriptors for this item: "Electronic Circuits," "Machine Tools," and "Servomechanisms," the latter two not being available to the machine system.

Figures 1–2 illustrate the present format for computer printout, with assignments in order of decreasing selection scores (numeric data to the right of descriptor names) for three different weighting formulas. These figures also illustrate differences in order of machine selection and of actual descriptors chosen by the

§ This is apparently due to the fact that although only each item's own title was source material, the weighting formulas to emphasize significance of words in the self-titles changed the effective proportion of the new material in the cited titles.

1 000191294550 PAPERS ON THE APT LANGUAGE

DESCRIPTOR NAME	1/N
MATHEMATICAL COMPUTER DATA	556
AUTOMATIC	510
LANGUAGE	487
SCIENTIFIC RESEARCH	330
HUMAN ENGINEERING	326
COMMUNICATION SYSTEMS	310
COMMUNICATIONS THEORY	300
MACHINE TRANSLATION	257
DATA TRANSMISSION SYSTEMS	247
INFORMATION THEORY	246
MATHEMATICAL LOGIC	244
SCIENTIFIC REPORTS	240

DESCRIPTOR NAME	1/TN.O
LANGUAGE	2356
AUTOMATIC	1589
MACHINE TRANSLATION	1460
PROGRAMMING	1212
MATHEMATICAL COMPUTER DATA	1111
DIGITAL COMPUTERS	1001
MATHEMATICAL LOGIC	928
COMPUTERS	918
DATA STORAGE SYSTEMS	915
DISPLAY SYSTEMS	765
CODING	643
COMMUNICATIONS THEORY	610

DESCRIPTOR NAME	1/N.O
✓LANGUAGE DYC, AFP, TCB	5078
✓MACHINE TRANSLATION AFP	3114
AUTOMATIC WWY, AFP, TCB	2652
✓PROGRAMMING DYC, WWY, AFP, TCB	2603
DIGITAL COMPUTERS WWY, TCB	2337
DATA STORAGE SYSTEMS	1952
COMPUTERS DYC, WWY, AFP, TCB	1906
MATHEMATICAL LOGIC	1769
✓DATA PROCESSING SYSTEMS WWY, AFP, TCB	1542
MATHEMATICAL COMPUTER DATA	1529
CODING DYC, AFP,	1239
DISPLAY SYSTEMS	1139

Figure 1.

1 000441294550 PAPERS ON THE APT LANGUAGE

DESCRIPTOR NAME	1/N
AUTOMATIC	3365
MATHEMATICAL COMPUTER DATA	3058
✓LANGUAGE	2473
✓PROGRAMMING	2375
DESIGN	1889
CYBERNETICS	1816
HUMAN ENGINEERING	1519
COMMUNICATIONS THEORY	1500
TRACKING	1265
COMPUTER LOGIC	1212
SCIENTIFIC REPORTS	1205
✓MACHINE TRANSLATION	1020

DESCRIPTOR NAME	1/TN.O
✓LANGUAGE	9819
✓PROGRAMMING	7169
✓MACHINE TRANSLATION	6840
AUTOMATIC	6372
MATHEMATICAL COMPUTER DATA	6262
DESIGN	5472
DIGITAL COMPUTERS	3366
COMMUNICATIONS THEORY	3069
COMPUTERS	2950
DATA STORAGE SYSTEMS	2790
MATHEMATICAL LOGIC	2730
SCIENTIFIC REPORTS	2457

DESCRIPTOR NAME	1/N.O
✓LANGUAGE	21251
✓PROGRAMMING	14929
✓MACHINE TRANSLATION	14535
AUTOMATIC	12164
DESIGN	11828
MATHEMATICAL COMPUTER DATA	8618
DIGITAL COMPUTERS	7524
COMPUTERS	6018
DATA STORAGE SYSTEMS	5850
✓DATA PROCESSING SYSTEMS	5611
MATHEMATICAL LOGIC	5082
COMMUNICATIONS THEORY	4950

Figure 2.

SADSACT technique using cited titles as against abstracts as input.

In our earlier tests, the same items had been run both with cited titles and with abstracts against the teaching sample in which they had been included (that is, as both partially and entirely source material items). For these items, the 64.8% hit accuracy for the cited title case improved to 75.0% average accuracy in the abstracts case. However, when the new test items were run both ways against teaching sample #2, there was no significant difference in the average accuracies obtained as between using titles-and-cited-titles only and using titles-and-abstracts from the same items.

Problem of Evaluation

The problems of evaluation of any indexing method whatsoever, whether carried out by man or by machine (or by men aided by machines) are crucial. O'Connor [12] has aptly pointed out that: "... the question, 'Is mechanized indexing possible?' by itself is not answerable, because it is not intelligible. More positively expressed, if you want to ask clearly about the possibility of mechanized indexing, you should have both some notion of 'good retrieval,' and some notion of how good you want the retrieval to be which is permitted by mechanized indexing."

In advance of evaluations based on retrieval tests, we have relied, first, on comparisons with the prior human indexing. We have been faced with the mechanics of assuring consistency-of-indexing criteria as necessary for any realistic assessment of machine as compared to manual indexing of the same items. As we have noted, this is not possible even for "training" and "test" items which are all documents of interest in our collection and which were all indexed in approximately the same time period by the same DDC (ASTIA) indexer-staff. As between "teaching sample #1," "teaching sample #2" (of 100 and 103 items, respectively, with 70% overlap), and of test items "new" to both these samples there are obvious differences as to the descriptor-vocabulary and indexing philosophy. This type of discrepancy was even more marked in the first run of eight (1963) items against the (1960) basis of teaching sample #1.

More serious, however, are the questions of the O'Connor remarks previously cited: those of evaluation of any indexing method whatsoever. We have, in our test data, a provocative case in point. In Table 4 we show the machine-assigned descriptors for the paper by L. G. Roberts of M.I.T. entitled "Pattern recognition with an adaptive network." Although exactly 12 descriptors were required, by our present

TABLE 4. Descriptors assigned by computer to "Pattern Recognition with an Adaptive Network"

Data
Sampling
Learning
Simulation
Computers
Data Processing Systems
Design
Digital Computers
Mathematical Prediction
Mathematical Logic
Digital Systems
Mathematical Computer Data

program, to be assigned, none of the 12 so assigned by machine is entirely irrelevant or inappropriate. As it happens, this same document was indexed twice by DDC personnel, first as an unpublished report, later as a reprint of the version appearing in the open literature. Indexer A assigned to this item the four descriptors: "Coding," "Digital Computers," "Mathematical Computer Data," and "Programming." Indexer B assigned: "Digital Computers," "Electrical Networks," "Identification Systems," "Programming," "Simulation," and "Test Methods."

It is of interest, first, that although our machine method "hit" score was 50% with respect to Indexer A, and only 33.3% with respect to Indexer B, at the same time A's agreement with B was exactly 33.3% and B's with A was 50%. More significantly, however, the machine method was the only one of the three which assigned the crucial descriptor "Learning" (reflecting the self-organizing and adaptive features of the mechanism described) to the item. Indexer B assigned two *in*appropriate descriptors—"Test Methods" and "Electrical Networks," but the SADSACT technique did not make either mistake, even though the word "network" did appear in the item's own title.

A second problem of evaluation we are faced with involves the consequences of an assumption that the previously-given human indexing decisions, for a sampling of our collection, are in fact "good enough." This is a misleading assumption. As a case in point, of the approximately 200 items handled either in the teaching samples or as "new" items, only one could be appropriately indexed under the descriptor "Computers" that also involved analog computers or systems. Thus, for all but one of these items, if "Computers" was a relevant descriptor, so also was the descriptor "Digital Computers." There is considerable debate about claims for "consistency" of machine, as opposed to manual, methods of indexing. In some of our tests, for example, we assigned both the descriptor "Computer" and the descriptor "Digital Computers" in 41.2% of the cases in which either was assigned, but the human indexers assigned both if either was used only 33.3% of the time.

In general, very little data are available on interindexer consistency or on the likelihood that the same indexer would index the same item the same way on different occasions. Painter [13] reports figures for interindexer consistency as ranging from 54% similarity of indexing to 82%. Her data for the case of DDC show that: "One hundred and eleven starred descriptors were used the first time which did not appear the second (41%) and 98 starred descriptors were used the second time and not the first (36%)." Jacoby and Slamecka[7] report even less favorable figures, with interindexer consistency as low as 18% in some cases. In the light of such indications, the automatic assignment indexing techniques appear promising. In addition, however, we have initiated the investigation of relevance judgments on the part of subject matter specialists reviewing the full texts of our test items.

Evaluation by Relevance Judgments

The data shown in Tables 1–3 compare machine assignments with those previously made by human indexers. An alternative method of evaluating the test results is to determine the

TABLE 5. Average per-cent agreement, one or more people with machine assignments

No. of Indexers	No. of Items	No. of Descriptors Assigned				
		12	9	6	3	1
2	1	50.0	66.7	66.7	100.0	100.0
3	4	47.9	52.7	58.3	66.7	75.0
4	10	40.0	45.7	55.6	53.3	60.0
5	10	54.2	58.9	68.3	80.0	90.0
	OVERALL	47.4	52.9	61.6	67.9	76.0

probable relevancy of those descriptors that are assigned by the machine. Twenty-five of the items in the test runs have therefore been submitted to one or more members of our staff, all of whom are users of the collection. They were asked to choose 12 descriptors for each item exclusively from the list of descriptors actually available to the machine. The results of these evaluations are shown in Table 5, which gives the summary data for judgments of relevance by one or more persons (including the initial DDC indexer) of descriptors assigned by the SADSACT technique. It should be noted that these judgments are based upon the independent assignment by the human analysts of these same descriptors to the same item, rather than by review of the machine assignments. Thus the results are probably conservative.

The results in Table 5 also show the percentages of machine-assigned descriptors judged by human analysts to be relevant for cases where the machine assigns fewer than 12 descriptors. It is evident that the fewer descriptors assigned by machine, the better the chance that human evaluators will judge those assigned to be relevant. It is also evident that the more the people who index, the better is the chance that one or more will assign a descriptor that the SADSACT model also assigns.

Tables 6 and 7 provide more detailed data for the cases where independent evaluations were made by several people. Table 6 shows machine-indexer and interindexer agreement for the 10 cases where four of our own "customers" had analyzed the items. It provides, for each of the different descriptors assigned by machine at the 6-descriptor cutting level, indication of the item for which these assignments occurred, and the agreement with assignments of that descriptor. Table 7 gives, for 20 cases in which four or more people made independent evaluations, the chances that the machine will choose descriptors also chosen by people for the first descriptor assigned, first two, . . . up to the total of 12.

Taking both the "hit" accuracy and the human agreement data into account, the SADSACT results compare favorably with those reported by Maron[9] (51.8%) and by Borko[3] in an experiment using the same computer literature abstracts as had been used by Maron (46.5%). Our results to date also appear to fall within the range of agreement-data for human interindexer consistency.

Future Considerations

Our present plans call, first, for continued testing and evaluation of a number of new items, varying such parameters as the size of the teaching sample, the number of descriptors to be assigned, and weights for both prior-association occurrences and descriptor selection. This additional experimentation can be carried out using the present preliminary program. Depending on the results of continued testing, we except to explore applicability of the method to other subject matter areas (both for more homogeneous and more heterogeneous collections) if they are favorable, and to study means for improvement by various means of reintroducing human judgment into the process, if the results are discouraging.

If the promise of the early results is substantiated on further runs, we shall also wish to re-investigate and refine the programming techniques. For our preliminary evaluation, many *ad hoc* choices and arbitrary assumptions were made. These will require modification. Additional programming refinements should provide for consolidated vocabulary-descriptor associations for larger teaching samples, incorporation of occurrences of n-tuples of significant words as "potential descriptors," experimentation with various weighting formulas, and the like.

The SADSACT indexing procedure differs from most other efforts at realization of assignment-type indexing by machine in two principal respects. First is the emphasis on *a posteriori,* completely mechanizable, techniques that nevertheless are based, in a sampling sense, upon prior judgments of people with respect to specific requirements of real job-situations. The second differentiating aspect is the insistence not only upon obtaining the consensus of the collection as the basis of vocabulary compilation and word-descriptor associations, but also of going beyond the individualistic usage of language and terminology by a particular author. This latter emphasis tends to reduce the effect of linguistic idosyncracies and consequent scatter of like items under a number of different indexing terms.

TABLE 6. Agreements, by descriptor and item, of one or more persons with first six descriptors assigned by machine to each of 10 test items

Descriptor	Number of persons agreeing with machine assignment						% Agreement Assignments of this Descriptor
	5	4	3	2	1	0	
Coding	CCCS*, LRBE, CPLT		TTDI				100.0
Theory	SHT			CCCS, CPLT	TTDI	WLI, SCC	66.7
Errors		CCCS, LRBE					100.0
Data Transmission Systems			CCCS	LRBE	SCC		100.0
Electronic Circuits						CCCS, LRBE	0.0
Information Theory	TTDI			CCCS			100.0
Language		WLI, CPLT, PAL					100.0
Machine Translation			CP LT	PAL		WLI	66.7
Data Processing Systems		MTSA		ASDS			66.7
Intelligibility		WLI					100.0
Bibliography						WLI, TTDI	0.0
Design			MTSA	ASDS		SHT	66.7
Human Engineering	MTSA				SHT		100.0
Automatic			PAL	MTSA			100.0
Synthesis						LRBE	0.0
Classification						LRBE	0.0

Data Storage Systems				ASDS, PAL MTSA	0.0
Data		ASDS			100.0
Mathematical Logic					0.0
Communications Theory		SCC			50.0
Programming	CPLT, PAL				100.0
Digital Computers		CPLT	PAL, SHT		100.0
Probability			TTDI	SCC	100.0
Reasoning	SHT			TTDI	50.0
Computers	SHT				100.0
Circuits				SCC	100.0
Tracking				SCC	0.0
Documentation				MTSA	0.0

*Note: Items are identified by initials of first 4 significant words in titles, see Tables 1-3.

TABLE 7. Number of descriptors assigned by machine, also chosen by one or more human indexers

Item	1	2	3	4	5	6	7	8	9	10	11	12
CCCS	1	2	3	4	4	5	6	6	7	7	7	7
ICC	1	2	2	3	4	5	6	7	8	8	8	8
UCCT	1	2	2	3	4	5	5	5	5	5	5	5
MCL	1	2	3	4	5	5	5	5	5	6	6	6
WLI	1	1	1	1	2	2	3	3	4	4	5	5
MCGE	0	1	1	2	3	4	5	5	6	7	7	7
MTSA	1	2	3	4	4	4	5	5	5	5	5	5
IAM	1	2	2	2	3	3	3	3	3	3	4	5
LRBE	0	0	1	2	3	3	4	4	4	5	5	6
ASDS	1	1	2	2	2	3	4	5	5	5	5	6
CPLT	1	2	3	4	5	6	6	6	6	6	6	6
SRPC	0	1	2	2	2	3	3	3	3	3	4	4
RPSN	0	0	0	1	1	2	2	3	3	4	4	4
PAL	1	2	3	4	5	5	6	6	7	7	8	8
TTDI	1	2	3	3	3	4	4	4	4	5	6	7
PMSO	1	1	1	1	1	1	1	1	1	1	1	1
HPFW	1	1	1	1	1	1	2	2	2	2	2	2
SHT	1	2	3	4	4	4	4	5	5	6	7	7
SCC	1	2	3	4	4	4	4	5	5	6	7	7
CIOP	0	1	2	3	4	5	6	6	6	6	6	6
Average Per-cent Chance that people will agree	75.0	70.0	66.7	66.3	64.0	62.5	60.7	56.2	52.8	51.0	49.1	47.1

It also tends to produce a normalizing effect so that the various items in the collection are indexed more consistently than is possible if each item is processed as an entirely independent and self-sufficient entity.

The word "training" as applied to our proposed method for automatic indexing is somewhat of a misnomer in terms of our actual experiments to date. It is justified in the sense that a training sequence, using a "teaching sample," gives the machine a representative set of "acceptable" input-to-desired-output examples. It is not justified, as yet, in the sense that the system as presently programmed could *significantly* modify or adapt its procedures, without human intervention, to new vocabulary or subject emphases in the input items. There is reason to suppose, however, that the method can be adapted to new and changing vocabularies (both of "descriptors" and of author terminology) by throwing out for human inspection: occurrences of words not on the general stop-list, yet not in the vocabulary; and occurrences of non-purged words in title-and-citations coinciding with the name of a descriptor, when that descriptor has not been assigned (e.g., an early occurrence of the word "plasma" in a physics paper, but not involving enough other words previously associated with blood chemistry, hematology, and the like, to assign physiological or medical field descriptors).

If this proves feasible, potentialities of

training the machine system to recognize new descriptor-assignment possibilities, to sharpen its own recognition-selection-assignment processes in terms of current real-life input, and to recognize and display anomalies, changes, and trends, may supply significantly improved problem-solving capabilities in the changing world of literature search as aided by machine. Although the SADSACT results to date are quite limited, the procedure indicates some promise of being quick, relatively inexpensive and consistent, and it capitalizes on one-time key stroking of descriptive cataloging information available for multiple purposes.

REFERENCES

1. BAKER, F. B. "Information Retrieval Based upon Latent Class Analysis." *Journal of the Association for Computing Machinery*, 9:4 (October 1962) 512–521.
2. BAXENDALE, P. B. "Machine-Made Index for Technical Literature — An Experiment." IBM JOURNAL OF RESEARCH AND DEVELOPMENT, 2:4 (October 1958) 354–361.
3. BORKO, H. "The Construction of an Empirically Based Mathematically Derived Classification System." In PROCEEDINGS 1962 SPRING JOINT COMPUTER CONFERENCE, American Federation of Information Processing Societies, 1962, pp. 279–289.
4. BORKO, H., and M. D. BERNICK. "Automatic Document Classification." JOURNAL OF THE ASSOCIATION FOR COMPUTING MACHINERY, 10:2 (April 1963) 151–162.
5. EDMUNDSON, H. P., and WYLLYS, R. E. "Automatic Abstracting and Indexing—Survey and Recommendations." COMMUNICATIONS OF THE ACM, 4:5 (May 1961) 226–234.
6. HERNER, S. "Methods of Organizing Information for Storage and Searching." AMERICAN DOCUMENTATION, 13:1 (January 1962) 3–14.
7. JACOBY, J., and V. SLAMECKA. "Indexer Consistency under Minimal Conditions." Bethesda, Md., Documentation Inc., Nov. 1962 lv. Contract AF 30(602)2616, Project 4594: RADC TR 62–426: AD–288 087.
8. LUHN, H. P. "A Statistical Approach to Mechanized Encoding and Searching of Literary Information." IBM JOURNAL OF RESEARCH AND DEVELOPMENT, 1:4 (October 1957) 309–317.
9. MARON, M. E. "Automatic Indexing: An Experimental Inquiry." In MACHINE INDEXING, PROGRESS AND PROBLEMS, Washington, D. C., American U., 1961, pp. 236–265.
10. MARON, M. E., and J. L. KUHNS. "On Relevance, Probabilistic Indexing and Information Retrieval." JOURNAL OF THE ASSOCIATION FOR COMPUTING MACHINERY, 7:3 (July 1960) 216–244.
11. NEEDHAM, R. M. "A Method for Using Computers in Information Classification." In INFORMATION PROCESSING 1962 (Proceedings of IFIP Congress, Munich), Cicely M. Popplewell, Editor (Amsterdam, North-Holland Publishing Co.), pp. 284–287.
12. O'CONNOR, J. "Some Remarks on Mechanized Indexing and Some Small-Scale Empirical Results." In MACHINE INDEXING, PROGRESS AND PROBLEMS, Washington, D. C., American U., 1961, pp. 266–279.
13. PAINTER, A. F. "An Analysis of Duplication and Consistency of Subject Indexing Involved in Report Handling at the Office of Technical Services, U.S. Department of Commerce." Rutgers State U., Ph.D. Dissertation, 1963, p. 135. Washington, D. C., Office of Technical Services, PB 181501.
14. STEVENS, M. E. "Preliminary Results of a Small-Scale Experiment in Automatic Indexing." To be published in PROCEEDINGS, NATO Advanced Study Institute on Automatic Document Analysis, Venice, 1963.
15. SWANSON, D. R. "Automatic Indexing and Classification." Paper presented at the NATO Advanced Study Institute on Automatic Document Analysis, Venice, 1963.

EXPERIMENTS IN INFORMATION CORRELATION*

J. L. Kuhns and C. A. Montgomery
The Bunker-Ramo Corporation
Canoga Park, California

1. INTRODUCTORY REMARKS

It is reasonable to suppose that information systems of the future will have control over portions of a document as well as over full text. In addition, we suppose that many users will want output in order to verify hypotheses or to form new hypotheses. This paper is concerned with the facilitation of the second procedure. Thus the emphasis is not on the two classical problems of information systems, namely, not retrieving relevant material, or retrieving irrelevant material; but rather on the problem of assimilability of the output by the user.

The particular aspect of the assimilability problem which will concern us is the effect of information loads of varying sizes upon the user's ability to assimilate and correlate data. It seems plausible to assume that the user's ability to correlate information is a function of the volume of information to be assimilated; and moreover, that this ability will improve with increasing volume until a certain critical volume is reached, beyond which the load of information causes the user's correlation ability to deteriorate rapidly. This assumption gives rise to two fundamental questions:

1. What is this critical volume in a given situation?
2. Is it possible to structure information in some way so as to improve the user's ability to assimilate and correlate, thus extending the critical limit?

Experiments performed in connection with an earlier project on information retrieval and correlation provided some interesting preliminary answers to these questions.

The library for this series of experiments consisted of a simulated intelligence file of sentence fragments developed from a book—"Kogun—The History of the Japanese Army in the Pacific." A machine-produced concordance of all words contained in the fragments served as an index to the file.

The experiment of particular significance here is one which dealt specifically with the time dimension. All fragments containing any reference to time of day (for example, "noon," "0645," "dawn," "8:00 a.m.") were retrieved. About 25 fragment cards were retrieved in all. The fragments of this set had at least two attributes in common: they concerned the Pacific War in some way, and contained some reference to time of day.

The randomly ordered cards were presented to several persons, who noticed only that all cards referred to battles in the Pacific War. When the cards were reordered into two groups of night versus day, however, a striking correlation emerged—namely, that all daylight attacks had been initiated by U.S. forces and all

* The work reported in this article was supported by the National Aeronautics and Space Administration under Contract NASw-538.

night attacks by the Japanese. (This correct inference, incidentally, was nowhere explicitly stated in the book.)

The results of this experiment suggest that the critical volume of information may in fact be extremely small. Only 25 cards were involved, and each card contained a single fragment, which in this case was a minimal piece of information. Moreover, these fragments had certain attributes in common—"Pacific War" and "time of day." The correlation described above, however, was not discovered until the set of fragments was structured in terms of conceptual subgroups—"night," "day"—included in the semantic range represented by "time of day."

These findings suggest that structuring of data can extend the critical limit on volume, and further, that the elements of this structure should be subsets consisting of closely related members of the main set. The results of the earlier experiment thus established a point of departure for the present investigation.

2. EXPERIMENTAL DESIGN

A considerably more complex series of experiments was designed to test whether the "Kogun" discoveries provided essentially correct answers to the two questions raised above. We decided to create a set of data structured along the lines described above—that is, a set whose structure is based on subsets consisting of closely related members of the main set. A second set of data more loosely structured than the first would be provided by the main set itself. A third set would consist of unstructured data compiled randomly. The ability to assimilate and correlate the information presented in these variously structured sets of data would then be tested by requiring experimental participants to list inferences suggested by each set. Based on the "Kogun" experiment, it was assumed that the first set of data, being the most highly structured, and therefore, most easily assimilated, would be most productive of inferences; and the random data, least productive. The scope of the initial experiments was limited to generating and testing the first set of data.

The type of structure represented by the first set of data is most conveniently generated by "clustering" techniques. The object is to cluster the data in natural groupings whose members are associated through co-occurrence of concepts. The procedure is as follows: if two items of the data set have a concept in common they are said to be associated. A cluster is then defined as a collection of items which are mutually associated as well as being maximal with respect to this property. This last condition means that an item cannot be added to the collection without violating the property of mutual association. The definition of association can be strengthened by stipulating that several concepts must occur in common. The technique used is a mathematical procedure for exhaustively listing these clusters.

3. EXPERIMENTAL PROCEDURE

Materials to be used for the creation of the three data sets were a fragment file of sentences generated from a collection of documents on advanced propellants and a machine-produced concordance which served as an index to the file.

The first step in the experimental procedure consisted of selecting a single fragment called the F_1 fragment, which would serve as the

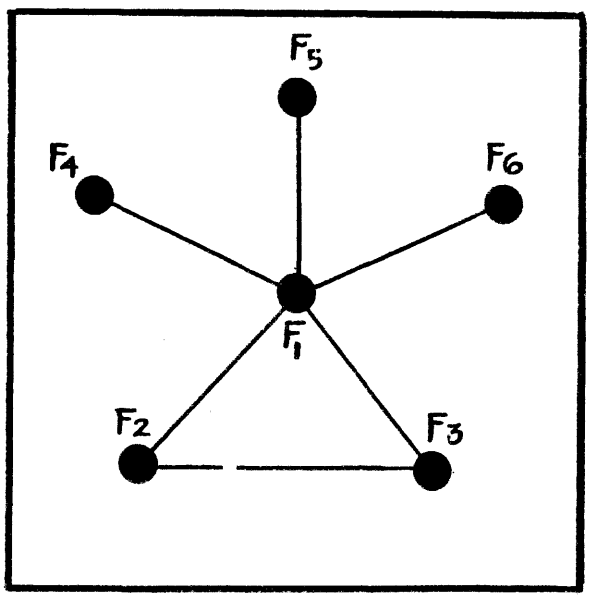


Figure 1. Illustration of F_1 star.

nucleus for a set consisting of all fragments related to the F_1 fragment through co-occurrence of concepts. This is diagrammed in Figure 1. The lines between fragments indicate the co-occurrence of concepts; for example, if F_1 contains the words "small motors" and F_2 contains the word "micro-burner," then the same concept can be said to occur. The set so formed is called the F_1 "star." Clusters of fragments associated by co-occurrence of concepts in addition to the F_1 defining concepts can then be derived from the F_1 star.

As an illustration, in Figure 1 the set $\{F_1, F_2, F_3\}$ is a cluster, so are $\{F_1, F_4\}$, $\{F_1, F_5\}$, $\{F_1, F_6\}$. These clusters can serve as the maximally structured set of data for the experiments, while the members of the F_1 star would constitute the loosely structured set.

Since the magnitude of the task of developing these structures manually is directly proportional to the number of fragments involved, it is essential that the membership of the star should not be too large. On the other hand, if its membership is too restricted, the star will not contain enough information for valid generalizations to be made. As the membership of the F_1 star depends on the number of concepts contained in the F_1 fragment and their distribution throughout the fragment file, it can be seen that selection of the F_1 fragment is the crucial step in the experimental procedure.

Attempts to limit the size of such stars by arbitrarily restricting the number of fragments taken from each document proved unsuccessful, as did several other artificial means of coping with an oversupply of related fragments. Many potential F_1 fragments were selected, found inappropriate, and rejected before the following F_1 fragment was chosen:

> "Space mission studies have indicated the need for small rocket motors of low thrust (of the order of 100 lb.) for attitude control, midcourse trajectory corrections, rendezvous, and so forth."

Concepts used to define the F_1 star were as follows:

small *and* motors	or	micro *and* motors	
burners			burners
engines			engines
space			
trajectory			
control *and* attitude			
rendezvous			

All cards containing these concepts were then retrieved from the file. There were 24 in all, yielding an F_1 star of 24 members, which proved to be a workable amount.

A list of concepts was then developed for each of these fragments and the list for each fragment was compared with that of all other fragments in the F_1 star to determine which fragments had the greatest number of concepts in common. A correlation matrix was created in which fragments having two or more concepts in common were labeled as strongly correlated, those having only one common concept as weakly correlated. "Strong" and "weak" clusters were then derived from the matrix by the mathematical technique mentioned above.

Since a greater degree of relatedness or structure is required by the defining criteria for strong versus weak clusters, these clusters represent legitimate materials for testing the user's ability to assimilate and correlate information having various degrees of structure. It was expected that inferences derived from strong clusters would exhibit some distinctive property as compared to weak clusters.

A further test of assimilation and correlation ability could be achieved by imbedding the clusters in successively larger segments of text, on the theory that correlations obvious in the clusters as represented by the fragment cards would be obscured if the clusters were instead represented by the documents from which they were culled. We therefore selected 12 clusters —6 strong and 6 weak—which were represented by four levels of material:

Level	*Description*
D	Cluster is represented by the fragment cards.
C	Cluster is represented by the sentence on the fragment card plus the six sentences (3 preceding, 3 following) which form its immediate context within the document.

B Cluster is represented by a machine-produced extract of the document.

A Cluster is represented by the document itself.

The clusters were then divided into four groups of three clusters each. The material for level A of the first group of clusters was placed in a folder marked for the first participant along with the level B material for the second group of clusters, the level C material for the third, and the level D material for the fourth, so that none of the four participants (specialists in the field of advanced propellants) would have the opportunity to draw inferences from the same cluster on two levels. These elaborate precautions were to prevent the participants from becoming familiar with the same cluster through its representations on various levels, causing a tendency to make the same inference for the different levels rather than looking for new correlations which might be exposed in proceeding to a level where the cluster is less deeply imbedded in peripheral material. The participants were instructed to begin with level A and proceed to level D, observing the time limits (given below) imposed for reading the material and making inferences:

Table of Maximum Times

Level	Reading Time	Time for Deriving Inferences
A	One-half hour	20 minutes
B	20 minutes	20 minutes
C	15 minutes	20 minutes
D	5 minutes	20 minutes

4. RESULTS

A total of 152 inferences were listed by the participants. Of these, exactly half were derived from strong clusters and half from weak. The level from which the largest number of inferences was derived was the A level, which also contained the most material. 51 inferences were listed for the A level, 42 for the B level, 34 for the C level and 25 for the D level, indicating a strong correlation between quantity of material and number of inferences produced, which we had intended to offset to some extent by imposing time limits for reading and inference listing. The S-16 cluster (representing level D) and the inferences derived from it are listed below.

S-16 Cluster (Level D)

This motor uses small quantities of propellants which can be made on a laboratory scale yet is large enough that the heat transfer losses are small.

The combustion gases are cooled to a much greater extent relative to that in a larger motor, thus lowering c (characteristic velocity).

Space mission studies have indicated the need for small rocket motors of low thrust (of the order of 100 lb.) for attitude control, midcourse trajectory corrections, rendezvous, and so forth.

Inferences for Cluster S-16

1. *Inferences at A-level*
   1. Propellant evaluations do not apear to be standardized.
2. *Inferences at B-level*
   1. Hypergolic rocket propellants permit optimum utility of small thrust engines.
   2. Injector configuration is a critical parameter in achieving maximum performance of small, low thrust rocket engines.
   3. High energy, toxic rocket propellants can be handled without hazard.
3. *Inferences at C-level*
   1. Propellants can deteriorate upon standing over long periods of time.
   2. Impurities may be the cause of uncontrolled burning and/or detonations and decrease performance of propellants.
   3. In small motors heat transfer through components lowers performance significantly.
   4. Better mixing of propellants increases performance.
   5. N_2O_2 and N_2H_4 are the best state-of-the-art combinations which satisfy requirements.
   6. Testing on small sub-scale is sufficient for accepting or rejecting a propellant combination for use on larger motor tests.

4. *Inferences at D-level*
    1. Low thrust, efficient, simple rocket motors have a variety of potential research and development applications.
    2. In a small motor, combustion gases are cooled to a greater extent than in a large motor probably because of the greater ratio of heat transfer surface to combustion chamber volume in the small motor.

The inferences were then uniquely classified into the following categories:

Propulsion Systems	Combustion Efficiency
Propellant Combinations	Combustion Stability
Propellant Evaluations	Ignition
Reliability	Heat Transfer
Performance	Problem Areas
Design	Extrapolation of Test Results
Chambers	Applications for Small Motors
Storage and Handling	

The inferences derived on the various levels were compared to determine whether the classes of inferences had members from all levels; or whether, as we had postulated, the relations between members of a cluster would be obscured when the cluster was represented by large segments of data, so that the same type of inference could not be derived from all representations of the same cluster. This assumption proved false in the present investigation, which might be attributed to the fact that the critical volume was not reached due to the overgenerous time limits. The same types of inference were frequently listed for levels A through D.

The original fragment cards making up the clusters were then assigned to as many of the above categories as were applicable in order to compare the extension of the clusters with that of the inferences derived from them. Intersections of categories were plotted for the clusters and for inferences made at the D level—i.e., those derived directly from the clusters themselves. A correlation coefficient was used to determine the degree of "overlap" between clusters and inferences, thus measuring their conceptual closeness. Although the correlation coefficients for both strong and weak clusters were good, inferences derived from strong clusters proved significantly more highly correlated with their parent clusters than did inferences derived from weak clusters. The strong clusters scored .61, the weak ones .43, where the coefficient is on a scale from —1 to +1 with 0 being the point of random correlation.

Although these data support to some extent the trends indicated by the "Kogun" experiments, they also raise some interesting questions which can only be answered by comparative data of the type outlined above in describing our design for these experiments. One of these questions arises from the fact that the participants were able to recognize relations between members of a cluster and make inferences based on these relations even when the cluster was represented by large segments of text. Would the same apply for more loosely structured sets of data—if the 24 members of the F_1 star, for example, were imbedded in successively larger segments of text, would the basic relation still be discernible, or would inferences derived be diffuse and general rather than highly correlated with the star from which they were derived? What sort of inferences would be derived by applying the layering technique to a random selection of fragments related only by coming from literature in the same field?

A second type of question is more thought-provoking, and less readily answerable. It departs from the notion of an inference as a measure of the ability to assimilate and correlate information and considers instead the nature of an inference. We have discussed the correlation betwen basic data and inferences derived from these data, but is it reasonable to expect such correlation to exist? Perhaps the mark of a good inference is that it exhibits little conceptual relatedness to the data from which it was derived. Again, what is a "good" inference? Should "good" be interpreted as "non-trivial"? How much information should be carried in an inference? A systematic approach to these problems is explored in the following section.

5. QUANTITATIVE EVALUATION OF INFERENCES

In developing a systematic procedure for evaluating inferences, it is convenient to think of the presented information set S as having three components: information content, volume, and structure. Let "s" denote the proposition expressed by the content of the set, and suppose that the analyst, after examining the set, produces the inferences (hypotheses, conjectures) $h_1, \ldots, h_n$. By varying the structure, or volume, or content of S, different collections of inferences will be generated. The problem is then to develop a comparative evaluation of the "goodness" of the two collections. In order to do this we consider a quantitative evaluation based on the notion of a subjective probability function or credibility function or betting function. Such functions are discussed in References 1, 2, 3. We denote this function by "B" (for "betting function" or "belief"). Thus B(h, s) is the analyst's evaluation of the degree of belief in the hypothesis h after considering the evidence s. B(h) is his degree of belief in the hypothesis h prior to considerings. We suppose that B obeys the laws of probability so that B gives the value of the betting quotient (ratio of amount bet to total stake) that should be offered for a bet on the hypothesis in question.* In addition, the quotient should be psychologically fair in the sense that the bettor must alternately be willing to bet on the negation of the hypothesis with the complementary quotient.

Let us suppose for the present that we actually can obtain three fundamental values B(h), B(h, s), and B(s). Since the "goodness" of a collection of inferences should in some way depend on their validity, novelty, dependency on s, and non-triviality, let us see how these things can be defined in terms of the B-function.

It seems appropriate to define the novelty of h to be $B(\bar{h})$, where $\bar{h}$ is the negation of h. Thus the novelty of h is taken to be

$$B(\bar{h}) = 1 - B(h)$$

i.e., the complement of the degree of belief in the inference before examining the supporting data. We define the validity of the inference to be B(h, s); i.e., the degree of belief after examining the supporting data. It appears that the "score" of an inference should increase with increasing validity and novelty and that the product of validity and novelty would therefore be a good choice for the score. On the other hand, an inference quite independent of s should score zero. Independence is characterized by the condition

$$B(h, s) = B(h)$$

so that the product of validity and novelty for an independent inference is $B(h) \cdot B(\bar{h})$. Thus, we propose to score inferences as the product of validity and novelty less the independence value of this product. That is to say, we take the final score to be

$$G(h, s) = [B(h, s) - B(h)] \cdot B(\bar{h})$$

The goodness of the inference is therefore measured as the product of the change in degree of belief and the novelty.

Let us now test this scoring schema for various extreme conditions: (1) h is tautologous (e.g., h is "A is A"); (2) h is contradictory (e.g., h is "A is not A"); (3) h is trivial (e.g., h = s); (4) s logically implies h (deductive inference); (5) h is independent of s (the supporting data does not relate to h). These five cases are shown in the table below. As well as showing the notions of validity and novelty, we also include the notions of "strength of supporting data" and "relative validity."

We now examine how the procedure would apply to some actual data. We use the "Kogun" experiment discussed above. The hypotheses to be considered are: (h_1) Japanese attack at night in a particular case; (h_2) Americans attack in day in a particular case; (h_3) Japanese almost always attack at night; (h_4) Americans almost always attack in day: (h_5) h_3 and h_4 (i.e., the assertion of the conjunction of h_3 and h_4). The supporting data s is that given by the 25 fragment cards.

Since the source of the data is from the book "Kogun" and consists of supposedly factual reports, let us assign a rather high value to B(s), say 0.9. Disregarding s, we consider each of the hypotheses h_1 and h_2 as quite plausible

*Thus betting schemes based on B will not lead to a net loss in every possible case. (See Reference 1, 3.)

TABLE OF EXTREME CONDITIONS

Notion	Definition	h is tautologous	h is contradictory	h is trivial (h = s)	s logically implies h	h is independent of s
Strength of Supporting Data	$B(s)$	$B(s)$	$B(s)$	$B(s)$	$B(s)$	$B(s)$
Relative Validity	$B(s \text{ and } h)$	$B(s)$	0	$B(s)$	$B(s)$	$B(s)\ B(h)$
Novelty	$B(\bar{h})$	0	1	$B(\bar{s})$	$B(\bar{h})$	$B(h)$
Validity	$B(h, s$	1	0	1	1	$B(h)$
(Novelty) x (Validity)	$B(h, s)B(\bar{h})$	0	0	$B(\bar{s})$	$B(\bar{h})$	$B(h)\ B(\bar{h})$
Score	$[B(h, s) - B(h)]\ B(\bar{h})$	0	0	$B^2(\bar{s})$	$B^2(\bar{h})$	0

and, therefore, assign a high *a priori* degree of belief to both, say 0.9. Thus, the novelty of each is 0.1. The analysis of the hypotheses h_3 and h_4 is more subtle. Presuming we know nothing about the Pacific War, we imagine that perhaps the specific military situation would mean that sometimes night is better for attack, sometimes day. On the other hand, perhaps the general properties of the military situations of the kind discussed are such that night is always better or day is always better for an attack. This second consideration makes us think that $B(h_3)$ is near 0.5 (because we cannot choose between night and day). The first consideration makes us think that $B(h_3)$ is less than 0.5. Therefore, we take $B(h_3) = 0.4$. Similarly $B(h_4) = 0.4$. The hypothesis h_5, however, is completely surprising, so we take $B(h_5) = 0.1$.

To estimate the validity, we note that h_1 and h_2 are contained in s. Thus, $B(h_1, s) = B(h_2, s) = 1$. For h_3 and h_4, we note that s expresses frequency of occurrence of certain situations and that the relative frequency of occurrence of these situations as described in h_3 and h_4 is unity. Thus, we take $B(h_3, s) = B(h_4, s) = 0.9$ (slightly less than unity). Similarly, we take $B(h_5, s)$ close to 0.9 but somewhat less, say 9.8.* The relative validity and scores can then be computed. The results are shown in the table below and are seen to be intuitively satisfactory.

TABLE OF KOGAN INFERENCES

Notion	Definition	h_1	h_2	h_3	h_4	h_5
Strength of supporting data	$B(s)$	.9	.9	.9	.9	.9
Relative Validity	$B(s \text{ and } h)$	.9	.9	.81	.81	.72
Novelty	$B(\bar{h})$	.1	.1	.6	.6	.9
Validity	$B(h, s)$	1.0	1.0	.9	.9	.8
(Novelty) x (Validity)	$B(h, s) \cdot B(\bar{h})$	.1	.1	.54	.54	.72
Score	$[B(h, s) - B(h)] B(\bar{h})$	.01	.01	.30	.30	.63

The problem of obtaining the B values is still open and it should be approached keeping the following considerations in mind.

1. The inferences obtained often contain words indicating qualitatively the degree of belief (e.g., "it appears that," "possibly," "probably," etc.); there is evidence that there is a comparative notion of belief susceptible of quantization.

2. There may be difficulty in obtaining an honest appraisal of $B(h)$ after the analyst has examined the data s. Possibly this can be avoided by using a referee to make the appraisals.

3. Comparative evaluations are desired and the quantitative evaluations are for this purpose. Thus consistency in the evaluations is required only to the extent of the comparative results.

6. CONCLUDING REMARKS

In this study, it was assumed:

a. that a user requirement in certain information systems is the obtaining of data to form hypotheses

b. that this ability to form hypotheses is dependent upon the assimilation and correlation of the data presented

c. that the ability to assimilate and correlate information is in turn dependent upon

*The normative condition, $B(h_3 \ \& \ h_4, s) = B(h_3, h_4 \ \& \ s)$. $B(h_4, s)$, is considered in this appraisal.

the volume and structure of the information presented.

A measure of assimilability called the critical volume was postulated based on the results of earlier information correlation experiments. An experimental design was developed to determine the relation between this measure and the degree of structure in the data presented. A mathematical technique was used to produce two sets of structured data. Assimilation of information represented by these data was then tested by requiring experimental participants to derive inferences from the data presented. Inferences derived from more highly structured materials proved more highly correlated with the basic data set.

Thus in the course of these experiments, methods for systematically structuring data and measuring assimilability were designed and tested on a small scale. In addition, a procedure for quantitatively evaluating hypotheses (i.e., the procedure for measuring assimilability) is proposed as a tool for a feedback improvement to the structuring procedure. Results of these experiments, although preliminary, are generally positive, and suggest realistic approaches to further studies of information correlation.

REFERENCES

1. CARNAP, R., "The Aim of Inductive Logic," in *Logic, Methodology and Philosophy of Science:* Proceedings of the 1960 International Congress, Stanford University Press, 1962.
2. POLYA, G., *Patterns of Plausible Inference,* Vol. II, Princeton University Press, 1954.
3. SAVAGE, L. J., "Bayesian Statistics," in *Recent Developments in Information and Decision Processes* (ed., Machol, R. E., and Gray, P.), The Macmillan Company, New York, 1962.

SOME FLEXIBLE INFORMATION RETRIEVAL SYSTEMS USING STRUCTURE MATCHING PROCEDURES*

Gerald Salton and Edward H. Sussenguth, Jr.
Computation Laboratory, Harvard University
Cambridge, Massachusetts

INTRODUCTION

The comparison between stored information identifications and requests for information is one of the principal tasks to be performed in automatic information retrieval. In so-called descriptor systems, where information is represented by sets of independent key words, this operation is relatively simple, since it consists of a comparison between the respective "vertors" of key words. In many retrieval systems it has been found necessary or expedient to use more complicated constructs for the identification of information. Notably "role" indicators are often added to identify various types of key words, and "links" specify a variety of relations between key words. A complete identification for a document or an item of information is often represented by a graph, consisting of nodes and branches between nodes, to identify respectively the key words and relations between key words.[1,2,3] The matching of such information graphs with graphs representing requests for information is a relatively complicated and time consuming operation, particularly since the request structure can be made to match the information structure only partially and incompletely.

The graph matching problem arises also in document retrieval systems where certain significant portions of text are extracted and compared with the search requests. In such cases, it is possible to represent the syntactic structure of the text excerpts by abstract trees, and a tree matching procedure becomes necessary to compare the extracted information with the requests.[4,5] As before, an exact matching procedure would not be very helpful, since many different ways can be found to express the same ideas or requests. What is needed instead, is a procedure which permits inclusion of partly unspecified information, and which provides for the possible relaxation of the various conditions that render a complete match impossible at any given time. Graph matching techniques are, of course, also applicable to the comparison of items of information which exhibit inherently a multi-dimensional structure, such as electrical or pipeline networks, street or geographical maps, chemical molecular structures, and so on.

Extensive experience has been gained in the past with structure matching programs which operate on a "node-by-node" or a "piece-by-piece" basis.[6,7,8] In the node-by-node approach, the nodes of the two structures are compared one at a time, until either the complete structures match, or else an incompatibility arises; in the latter case it becomes necessary to backtrack to a point where there is agreement and try again with different elements. In the piece-by-piece approach, a dictionary of basic substructures is used to break a given structure into pieces which are then matched as a whole.

* This study was supported in part by the National Science Foundation under Grant GN-82.

Neither of the two techniques works well for any but the simplest structures. The node-by-node method usually requires extensive backtracking, involving the comparison of many hundreds of nodes for even very simple structures. The piece-by-piece approach, on the other hand, suffers from the fact that no standard, well-defined method exists for breaking a given structure into substructures.

A topological structure-matching procedure has been programmed for the 7090 computer which does not depend on a specific ordering of the nodes, nor on the presence or absence of certain specified substructures. Little or no backtracking is required, and the method can be used to detect complete as well as partial matches. The basic idea is to determine certain simple properties of the nodes of the two structures to be matched, and to equate those subsets of the nodes in the two structures which exhibit equivalent properties. A standard procedure is then used to form new matching subsets, and to break down already existing subsets into sets with fewer members. The procedure is completely determinate except in cases where it is necessary to resolve certain symmetries in the connection pattern of the nodes; in that case a guess (assignment) is made as to the correct solution; such a guess may later prove to have been right or wrong, and if wrong, may require some backtracking. In most practical problems, however, little backtracking seems to be needed.† Computer experiments indicate that the topological procedure is much more efficient than either the "node-by-node" or the "piece-by-piece" approach. Operating automatic retrieval systems based on the use of relatively complex structures (as opposed to sets of unconnected key words) seem therefore to become a practical possibility instead of merely a theoretically desirable goal.

An example is given first to illustrate the partial matching procedure, as well as the methods which may be used to alter one or both of the structures to be compared in order to make a match between them more likely. The procedure is then applied to the matching of document graphs with request graphs, and to a retrieval system based on the comparison of syntactically analyzed document excerpts with a stored phrase dictionary.

† A related strategy has been used by Unger to detect complete, rather than partial, isomorphisms between directed graphs.[9]

THE STRUCTURE MATCHING PROCEDURE‡

Consider first the problem of determining whether the graph of Fig. 1(b) is contained in the graph of Fig. 1(a), that is whether Fig. 1(b) is a subgraph§ of Fig. 1(a). The nodes in the two structures are labelled arbitrarily from ① to ⑨ and from ⓐ to ⓒ respectively, and the connection pattern of the nodes is represented by the binary connection matrices§ shown in Figs. 2(a) and 2(b). Since no additional information is furnished about either the nodes or the branches of the two graphs under consideration, all relevant properties of these graphs are in fact derivable from the con-

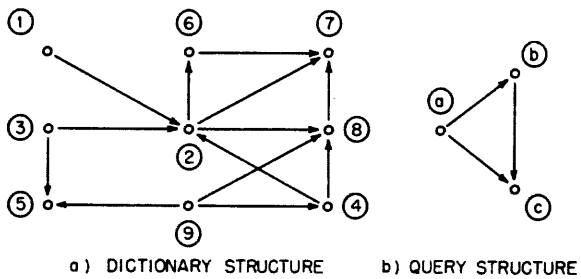


Figure 1. Directed Graphs.

‡ Theoretical foundations and proofs of convergence of the method are given in detail in Reference 10. The theory as well as applications to chemistry are also more fully treated in References 11 and 12.

§ A *graph* G consists of a set X (the nodes) and a set of relations between certain pairs of nodes (the branches). A matrix C such that $C_j^i = 1$ whenever there is a branch from node x_i to node x_j, and is 0 otherwise, is called the *connection matrix* of graph G. A *subgraph* H of G is obtained by removing from G certain nodes as well as all branches adjacent to the removed nodes. A *partial graph* J of G is obtained by removing from G some of its branches. A *partial subgraph* K of G is a subgraph of a partial graph of G. A *completed partial subgraph* L of G is a partial subgraph to which branches are added so as to preserve all original paths between the nodes included in L; specifically, if node z is removed from G in forming L and if there exist paths from x to z and from z to y in G, then a path exists from x to y in L for all x, y included in L.

	1	2	3	4	5	6	7	8	9
1		1							
2						1	1	1	
3		1			1				
4		1						1	
5									
6							1		
7									
8							1		
9				1	1			1	

	a	b	c
a		1	1
b			1
c			

a) DICTIONARY STRUCTURE b) QUERY STRUCTURE

Figure 2. Connection Matrices for Graphs of Figure 1.

nection matrices. The computer program is therefore based on the manipulation of binary matrices of the type shown in Fig. 2.

The heart of the algorithm consists in using various properties of the nodes and/or branches of the graphs in order to generate pairs of sets which must match if the two graphs are eventually to match. The following properties are particularly useful for this purpose:

a. the k^{th} order outward (or inward) degree of the nodes, that is the number of nodes reachable from a given node by outgoing (or incoming) paths of length k;

b. labels or identifiers which may be associated with nodes or branches;

c. the connectivity patterns of sets of nodes, that is, the nodes reachable from a given set of nodes by paths of length k.

The procedure for the graphs of Fig. 1 is outlined in Fig. 3. Only connections of length 1 have been used to simplify the exposition.

Set Number	Criterion for Set Formation	Corresponding Sets
I	Outward degree 1	{b} ⊆ {1,2,3,4,6,8,9}
II	Outward degree 2	{a} ⊆ {2,3,4,9}
III	Inward degree 1	{b} ⊆ {2,4,5,6,7,8}
IV	Inward degree 2	{c} ⊆ {2,5,7,8}
V	Partition	{a} ⊆ {2,3,4,9}
VI		{b} ⊆ {2,4,6,8}
VII		{c} ⊆ {2,5,7,8}
VIII	Outward connection, sets V	{b,c} ⊆ {2,4,5,6,7,8}
IX	sets VI	{c} ⊆ {2,6,7,8}
X	Inward connection, sets VI	{a} ⊆ {1,2,3,4,9}
XI	sets VII	{a,b} ⊆ {1,2,3,4,6,8,9}
XII	Partition	{a} ⊆ {2,3,4,9}
XIII		{b} ⊆ {2,4,6,8}
XIV		{c} ⊆ {2,7,8}
XV	Inward connection, sets XIV	{a,b} ⊆ {1,2,3,4,6,8,9}
XVI	Partition	{a} ⊆ {2,3,4,9}
XVII		{b} ⊆ {2,4,6,8}
XVIII		{c} ⊆ {2,7,8}
XIX	Assignment	{a} = {2}
XX	Outward connection, sets XIX	{b,c} ⊆ {6,7,8}
XXI	Partition	{a} = {2}
XXII		{b} ⊆ {6,8}
XXIII		{c} ⊆ {7,8}

Figure 3a. Correspondences Formed for the Graphs of Figure 1.

Set Number	Criterion for Set Formation	Corresponding Sets
XIX′	Assignment	{a} = {3}
XX′	Outward connections, sets XIX′	{b,c} = {2,5}
XXI′	Partition	{a} = {3}
XXII′		{b} = {2}
XXIII′		{c} = –

Figure 3b. Set Correspondences Resulting from Improper Assignment.

The initial set correspondences are shown in Fig. 3(a), lines I to IV. Set II, for example, is constructed by noting that the set of all nodes of outward degree 2 in the query structure, must correspond to the set of all nodes having *at least* outward degree 2 in the dictionary structure. The only node of outward degree 2 in Fig. 1(b) is ⓐ ; there are four nodes in Fig. 1(a) that have outward degree 2 or greater; ⓐ must therefore correspond to either nodes ②, ③, ④ or ⑨.

At this point it is necessary to generate smaller sets from the ones shown on lines I to IV of Fig. 3(a). This is done by noting, for example, that the set of query nodes contained in both sets I and III of Fig. 3(a) can correspond only to dictionary nodes which are also contained in sets I and III (plus possibly in other sets). The only query node contained in both sets I and III is ⓑ ; in the dictionary structure, nodes ②, ④, ⑥, and ⑧ are both in sets I and III, so that set {b} must be contained in sets {2, 4, 6, 8}. The set of possible correspondents of node ⓑ has then been reduced from the seven nodes of set I to the four nodes of set VI. This "set partitioning" procedure is performed by the computer by comparing the columns of the binary matrices exhibited in

SETS	NODES									
	a	b	c	1	2	3	4	5	6	7	8	9
I	0	1	0	1	1	1	1	0	1	0	1	1
II	1	0	0	0	1	1	1	0	0	0	0	1
III	0	1	0	0	1	0	1	1	1	1	1	0
IV	0	0	1	0	1	0	0	1	0	1	1	0

Figure 4. Matrix Representation of Sets I–IV of Figure 3a.

Fig. 4. Node ⓑ, for example, has column vector 1010; the only nodes of the dictionary structure including the pattern 1010 are nodes ②, ④, ⑥ and ⑧ with vectors 1111, 1110, 1010, 1011 respectively.

The partitioning process yields three new sets labelled V, VI and VII. New sets (VIII to XI) are added using outward and inward connections of length 1 from the sets V, VI, and VII. (The outward connection of the set {c} is empty, and therefore it is not included in the table.) The partitioning process is repeated, yielding sets XII, XIII, and XIV. Since the possible correspondents of node ⓒ have changed, it is not redundant to test the connectivity again. When this is done and another partition performed, sets XVI, XVII, and XVIII result. These pairs of sets are identical to those produced by the previous partitioning, and no simple properties of the nodes can be used at this point to generate new sets which would in turn result in a reduced partition.‖ An "assignment" is therefore made by postulating the correspondent for node ⓐ (set XIX). This assignment represents a guess which must later be verified for correctness. Partitioning of the sets XVI to XX yields the sets XXI to XXIII. New assignments (not shown in Fig. 3(a)) of ⓑ first to ⑥ and then to ⑧ then produce two one-to-one correspondences between the graphs of Fig. 1:

$$\left.\begin{array}{l} ⓐ \longleftrightarrow ② \\ ⓑ \longleftrightarrow ⑥ \\ ⓒ \longleftrightarrow ⑦ \end{array}\right\} \text{ and } \left\{\begin{array}{l} ⓐ \longleftrightarrow ② \\ ⓑ \longleftrightarrow ⑧ \\ ⓒ \longleftrightarrow ⑦ \end{array}\right.$$

If it is desired to obtain other possible correspondences, it is now necessary to go back to the sets XVI to XVIII and attempt other assignments for node ⓐ. Sets XIX′ to XXIII′ of Fig. 3(b) illustrate the assignment ⓐ ⟷ ③. This assignment yields a partition which is seen to be improper since node ⓒ cannot be included in the empty set. The assignment a ⟷ ③ is therefore not useful since it leads to an incompatibility. The other two possible assignments for node ⓐ, do, however, furnish acceptable one-to-one correspondences as follows:

$$\left.\begin{array}{l} ⓐ \longleftrightarrow ④ \\ ⓑ \longleftrightarrow ② \\ ⓒ \longleftrightarrow ⑧ \end{array}\right\} \text{ and } \left\{\begin{array}{l} ⓐ \longleftrightarrow ⑨ \\ ⓑ \longleftrightarrow ④ \\ ⓒ \longleftrightarrow ⑧ . \end{array}\right.$$

The four mappings obtained are easily verified by comparing Figs. 1(a) and 1(b).

A flowchart of the complete procedure is shown in Fig. 5. The procedure is seen to be

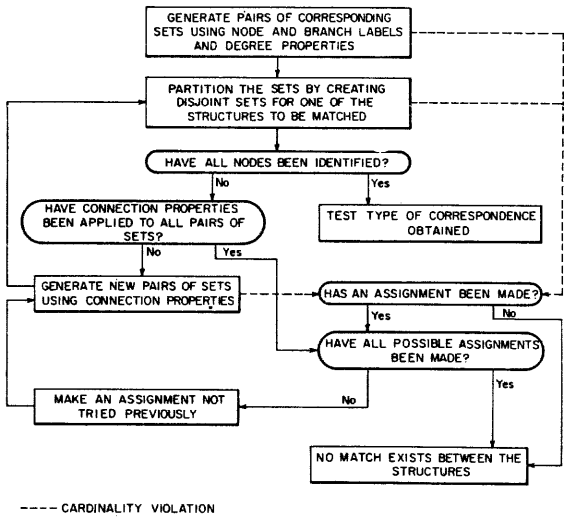


Figure 5. Simplified Diagram of Graph Matching Procedure.

iterative since the generation of corresponding sets is followed by a partitioning process, followed again by the generation of new sets, and so on. Alternate applications of partitioning followed by formation of new sets will result in one of three situations:

1. the membership of each set is reduced to one, thus exhibiting the complete match between the given structures;
2. an incompatibility arises between pairs of corresponding sets, that is, a cardinality

‖ The problem of choosing a "good" property set to be used for the generation of set correspondences is, in general, unsolved. For the example at hand, additional properties might, however, have been used. For instance, second order inward degrees yield the correspondence c — 2, 6, 7 after the first partition.

violation is found to exist between pairs of corresponding sets; in that case no match exists in general;

3. no incompatibility arises but repeated application of the partitioning procedure will not result in the formation of new sets; in that case, more than one match is generally possible, and it is necessary to perform an arbitrary assignment of correspondents for one of the nodes.

If a cardinality violation is detected, that is, if, for example, a set A is found to be included in a set B which has fewer members, as happened in the example for sets XXIII′, then the two structures being compared obviously cannot match. The comparison process can therefore be stopped immediately, unless the incompatibility resulted from a previous assignment; in the latter case, only that particular assignment can be discarded, and other possible assignments must be tried before deciding that the two structures do or do not match. The procedure to be followed in case of cardinality violation is shown by broken lines in Fig. 5. In practice, cardinality violations normally arise early for graphs which do not match, so that the procedure is very rapid in such cases.#

THE ADAPTIVE MATCHING PROCESS

In the example described in the preceding section, four isomorphisms were detected between the graph of Fig. 1(b) and that of Fig. 1(a). Clearly, it is possible to increase or decrease the number of matches (or, alternatively, to increase or decrease the probability of a match between any two given structures) by suitably relaxing or tightening the conditions which affect the matching process. If, for example, the unilateral connections (directed branches) in Fig. 1 are replaced by bilateral connections, and therefore the non-symmetric connection matrices of Fig. 2 are changed into the symmetric ones of Fig. 6, then eight additional isomorphisms will be found between the two graphs. In fact, three different isomorphisms will then exist between the "triangle" {a, b, c} and each of the triangles {2, 6, 7}, {2, 8, 7}, {4, 2, 8} and {9, 4, 8} included in the dictionary structure.

Non-matching graphs of fifty nodes required an average of less than one-half millisecond on the 7090 computer during a test run.

	1	2	3	4	5	6	7	8	9
1		1							
2	1		1	1		1	1	1	
3		1			1				
4		1						1	1
5			1						1
6		1					1		
7		1				1		1	
8		1		1			1		1
9				1	1			1	

a) DICTIONARY STRUCTURE

	a	b	c
a		1	1
b	1		1
c	1	1	

b) QUERY STRUCTURE

Figure 6. Symmetric Connection Matrices Derived from Graphs of Figure 1.

Another possible way of relaxing the conditions which are operative during the matching process is to permit the introduction between any two nodes in the query structure of a variable number of intermediate nodes. This process replaces the query structure of Fig. 1(b) by the new structures of Figs. 7(b) and 7(d). (The broken lines indicate indirect connections.) Since each of the intermediate nodes may or may not match a given node in the dictionary structure, it is now necessary to test whether the query structure is a completed partial subgraph (rather than a subgraph) of the dictionary structure. A comparison of Fig. 7(b) with the partial subgraph of Fig. 1(a) represented as Fig. 7(a), and a comparison of Fig. 7(d) with Fig. 7(c) reveals at least two additional completed partial subgraph matches that could be obtained in addition to the four subgraph matches already exhibited in the preceding section. Further completed partial subgraph matches not shown in Fig. 7 are also possible.

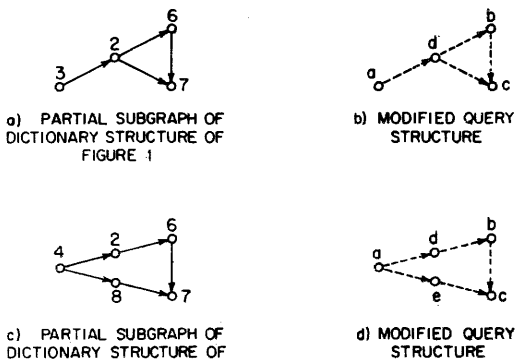


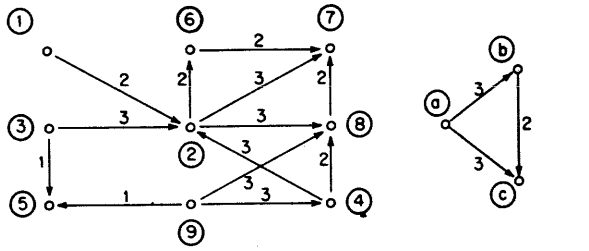
Figure 7. Matching Partial Subgraphs.

	1	2	3	4	5	6	7	8	9
1		1				1	1	1	
2						1	1	1	
3		1			1	1	1	1	
4		1				1	1	1	
5									
6						1			
7									
8						1			
9		1		1	1	1	1	1	

	a	b	c
a		1	1
b			1
c			

a) DICTIONARY STRUCTURE b) QUERY STRUCTURE

Figure 8. Path Matrices (Including Indirect Connections) for Graphs of Figure 1.



a) DICTIONARY STRUCTURE b) QUERY STRUCTURE

Figure 9. Abstract Graphs Including Branch Labels.

To determine whether a graph is a completed partial subgraph of another graph, it is no longer sufficient to know whether two nodes are directly connected or not, but it is also necessary to know whether a (possibly indirect) path exists between any pair of nodes. Thus the connection matrices of Fig. 1 must be replaced by the "path matrices" shown in Fig. 8 in which the i-jth element is 1 whenever a path exists from node i and to node j.** Each 1 in the matrices of Fig. 8 thus indicates either a direct or an indirect connection between the corresponding nodes, and use of the algorithm of Section 2 with the path matrices of Fig. 8 (instead of the connection matrices of Fig. 2) will generate the isomorphisms derived in Figs. 3 as well as a number of additional completed partial subgraph matches including those exhibited in Fig. 7.

Consider now, on the other hand, methods which will tighten the requirements to be met for a satisfactory match. Instead of specifying less information than for the directed subgraph comparison, it is now necessary to add restrictions to the graph of Fig. 1. A possible method consists in adding labels to the unlabelled branches of the graph to simulate, for example, various types of relations between the nodes. Another possibility is the addition of labels to the nodes of the graph so as to restrict the correspondents of a given labelled query node to only those nodes in the dictionary structure which carry the same label.

Consider first the two graphs shown in Fig. 9. Clearly these graphs are identical with those shown in Fig. 1, except for the added branch labels which distinguish three types of relations denoted respectively by the digits 1, 2, and 3. The binary connection matrix of Fig. 2 may now be replaced by a numeric branch label matrix as shown in Fig. 10, whose i-jth element

	1	2	3	4	5	6	7	8	9
1		2							
2						2	3	3	
3		3			1				
4		3						2	
5									
6							2		
7									
8							2		
9				3	1			3	

	a	b	c
a		3	3
b			2
c			

Figure 10. Branch Label Matrix for Graphs of Figure 9.

is n if there exists a branch of type n from node i to node j, and is 0 otherwise. In the set generation and partitioning procedure, it is then possible to keep with each node a list of branch labels of all outgoing (or incoming) branches,

Set Number	Criterion for Set Generation	Corresponding Sets
I	Outward degree 2 (or greater) and outward branch label (3,3)	{a} ⊆ {2,9}
II	Outward degree 1 (or greater) and branch label (2)	{b} ⊆ {1,2,4,6,8}
III	Inward degree 2 and branch label (2,3)	{c} ⊆ {2,7,8}
IV	Connections from set I including branch label (3)	{b,c} ⊆ {4,7,8}
V	Connections from set II and branch label (2)	{c} ⊆ {2,6,8,7}
VI	Connections into set III and branch label (2)	{b} ⊆ {1,6,8,4}
VII	Connections into set III and branch label (3)	{a} ⊆ {3,4,2,9}

Figure 11a. Set Correspondences for Graphs of Figure 9.

** The path matrix may be generated automatically as a sum of powers of the corresponding connection matrix.[13]

	a	b	c	1	2	3	4	5	6	7	8	9	
I	1	0	0	0	1	0	0	0	0	0	0	1	
II	0	1	0	1	1	0	1	0	1	0	1	0	{a} ⊆ {2,9}
II	0	0	1	0	1	0	0	0	0	1	1	0	{b} ⊆ {4,8}
V	0	1	1	0	0	0	1	0	0	1	1	0	{c} ⊆ {7,8}
V	0	0	1	0	1	0	0	0	1	1	1	0	
VI	0	1	0	1	0	0	1	0	1	0	1	0	
II	1	0	0	0	1	0	1	0	0	0	0	1	

Figure 11b. Set Inclusion Matrix for Sets of Figure 11a.

and, obviously, given a pair of corresponding sets not only must the nodes match as before, but the branch labels must match as well. The matching procedure is illustrated in Fig. 11.

The branch labels make it possible to generate a large number of sets at the outset. The set partitioning procedure illustrated by the set inclusion matrix of Fig. 11(b) then results in the formation of the three small sets reproduced in the figure. Assignment of ⓐ to either node ② or node ⑨ finally produces two one-to-one mappings as follows:

$$\left.\begin{array}{l} ⓐ \longleftrightarrow ⑨ \\ ⓑ \longleftrightarrow ④ \\ ⓒ \longleftrightarrow ⑧ \end{array}\right\} \text{ and } \left\{\begin{array}{l} ⓐ \longleftrightarrow ② \\ ⓑ \longleftrightarrow ⑧ \\ ⓒ \longleftrightarrow ⑦ \end{array}\right.$$

A comparison of the graphs of Fig. 9 can be used to verify that these two mappings are the only ones which obey the branch labelling restrictions.

As a last extension, consider now the two Syntol graphs[2] of Fig. 12. These graphs correspond to an actual document abstract and to a search request, respectively, as encoded under the Syntol system, and may be seen to be identical with the structures of Fig. 9 except for the added node labels. In order fully to represent the system, it is now necessary to add node label matrices to the connection matrices and

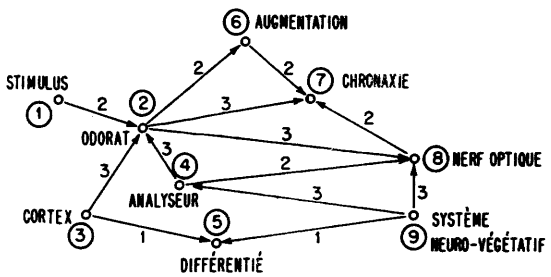


Figure 12a. Typical Syntol Document Graph.

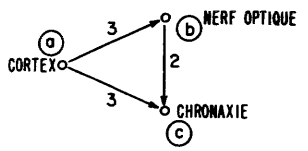


Figure 12b. Typical Syntol Query Graph.

to the branch label matrices. The node label matrices may be represented either as a table including all the node names together with the (possibly vacuous) corresponding labels, or alternatively as a full matrix whose i-jth element is 1, whenever label j is attached to node i. The node labels serve the same purpose as the branch labels, since they restrict the number of possible correspondents of a given node to only those nodes which either carry the same label, or else carry no label, thus indicating that they can match any node whatsoever that satisfies the remaining restrictions.

The procedure used to determine whether the query graph of Fig. 12 is a subgraph of the document graph is outlined in Fig. 13. Since all nodes are labelled, an immediate correspondence is established between the nodes of the two graphs under consideration (sets I, II and III of Fig. 13). It remains to determine whether the connections and branch labels are preserved. Sets IV of Fig. 13 reveal an incompatibility, since node ⓐ has two outgoing branches with a branch label (3), whereas the corresponding node ③ has only one such outgoing branch. Since a set containing two elements cannot be contained in a set containing only one element,

Set Number	Criterion for Set Generation	Corresponding Sets
I	Node label "cortex"	{a} ⊆ {3}
II	Node label "nerf optique"	{b} ⊆ {8}
III	Node label "chronaxie"	{c} ⊆ {7}
IV	Direct connections from set I with branch label (3)	{b,c} ⊆ {2}

Figure 13. Set Correspondences for Graphs of Figure 11 Using Direct Connections.

Set Number	Criterion for Set Generation	Corresponding Sets
I	Node label "cortex"	{a} ⊆ {3}
II	Node label "nerf optique"	{b} ⊆ {8}
III	Node label "chronaxie"	{c} ⊆ {7}
IV	Paths from I with branch label (3)	{b,c} ⊆ {2,7,8}
V	Paths from II with branch label (2)	{c} ⊆ {7}
VI	Paths into III with branch label (2)	{b} ⊆ {1,2,4,6,8}
VII	Paths into III with branch label (3)	{a} ⊆ {2,3,4,9}
VIII	Paths into II with branch label (3)	{a} ⊆ {2,3,4,9}

Figure 14. Set Correspondences for Graphs of Figure 11 Using Indirect Connections.

the subgraph test fails, and there is no need to proceed further.

It is therefore necessary to relax the matching conditions by taking into account indirect connections and intermediate nodes. The path matrix of Fig 8 is now used to verify that complete paths (rather than direct connections) and path labels are preserved by the correspondence in the node labels. The set correspondences, shown in Fig. 14, demonstrate that to each outgoing and incoming labelled path in the query structure there corresponds a path with similar properties in the document structure. The query graph therefore matches the document graph when indirect connections are taken into account.

An adjustable procedure for the comparison of query and dictionary structures can now be outlined. The process uses the same matching algorithm throughout, and is modified only by altering the matrices which represent the connection patterns and the branch or node labels. The exact strategy used in the progressive alteration of the matrices may be made to depend on the type of document collection being processed, and on preliminary retrieval tests. Clearly, the weaker the restrictions which affect the matching process, the more matches are likely to be obtained, and the larger therefore the collection of answers to a given search request.

In general, elimination of the branch labels from the query and document graphs reduces a variety of possible relations between terms to a single one (represented by an unlabelled branch). Replacement of directed by non-directed branches further reduces the ability to discriminate between a variety of relations, since a relation from A to B is now equivalent to one from B to A. Finally, removal of node labels simplifies both the search requests and the document identifications, since it eliminates from consideration some of the terms used as identifiers.

A possible strategy for the gradual broadening of matching criteria is as follows:

a. Use unmodified query structure Q and dictionary structure D and test whether Q is a subgraph of D;
b. If the preceding test is negative use path matrix including indirect connections to determine whether Q is a completed partial graph of D;
c. If the preceding test is again negative, selectively remove branch labels by altering branch label matrix and test again using first only direct connections (subgraph test), then indirect connections;
d. If matching conditions must be further relaxed, replace unilateral by bilateral connections and use symmetric connection matrices first with direct and then with indirect connections;
e. Finally, selectively remove node labels and test again for subgraph and then for incomplete partial graph.

A retrieval system using graph matching procedures in conjunction with natural language data is outlined in the next section.

A SENTENCE MATCHING PROCEDURE FOR DOCUMENT RETRIEVAL

A simplified automatic document retrieval system is shown in Fig. 15.[5,14] This system makes use of the standard statistical procedures, including the computation of word frequency counts, word associations based on co-occurrence in the same sentences or texts, document associations based on co-occurrence of words, and document relevance coefficients.[15,16] In addition, a dictionary or thesaurus may be used if available to normalize the vocabulary.

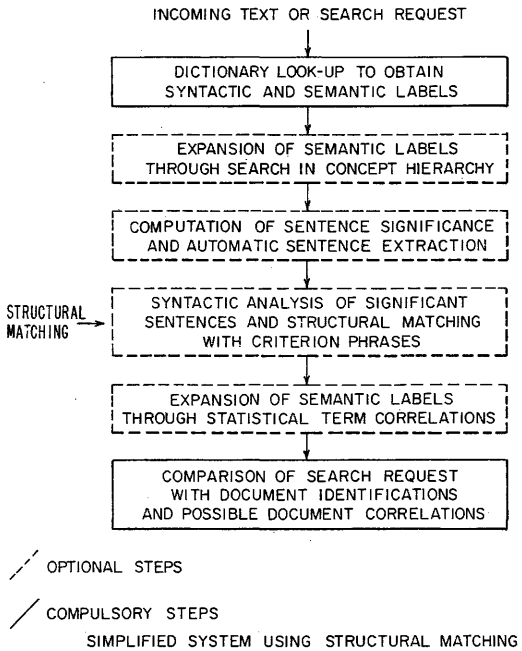


Figure 15. Simplified System Using Structural Matching.

The quantitative procedures may be supplemented by choosing a set of significant sentences, as determined by the statistical process, and using them to perform a structural analysis. Specifically, each word is furnished with one or more thesaurus category numbers (the semantic labels) as a result of the dictionary look-up procedure. If no dictionary is available, each word can of course function as a semantic label by itself. A syntactic analysis is then performed which determines a dependency structure for the words of a sentence, and also generates a syntactic label for each word. A typical dependency tree, resulting from an automatic syntactic analysis, is shown in Fig. 16. A syntactically analyzed sentence can of course be represented as before by direct and indirect connection matrices, as well as syntactic and semantic label matrices; moreover, these matrices can be generated automatically from the output furnished by the syntactic analysis program.[17]

It is now possible to compare the set of analyzed sentences or search requests with a set of "criterion phrases" included in a phrase dictionary. Each criterion phrase is representative of one or more subject categories, and if a match is obtained between a criterion phrase and an analyzed sentence or search request, the corresponding subject categories can be attached to the matching sentences or requests. To retrieve a set of documents in answer to a given search request, it is then sufficient to compare the subject categories attached to the requests, with the subject identifiers attached to the documents as outlined in Fig. 15.

A typical criterion phrase is shown in Fig. 17. Each criterion phrase is represented, as before, by an identification number and control

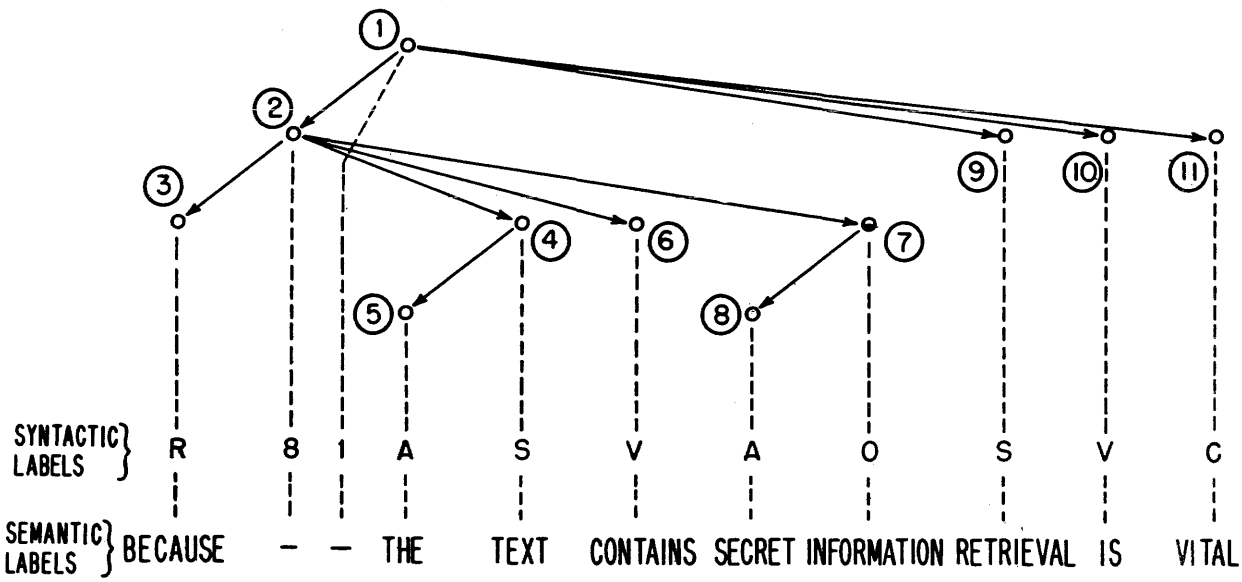


Figure 16. Typical Syntactic Dependency Tree.

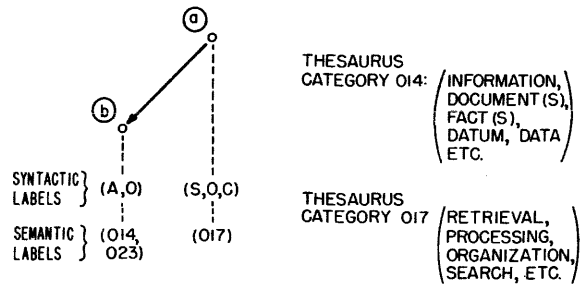


Figure 17. Typical Criterion Phrase.

information, the direct and indirect connection matrices, the syntactic and semantic node label matrices, and the category indicators which identify the subject classes for the given phrase. The semantic node labels attached to the sample criterion phrase are decoded in Fig. 17.

The matching process between a given criterion phrase and a sample sentence or search request is identical with that used in the preceding section for document graphs. That is, two principal criteria must be satisfied:

1. Given a specified node of the criterion phrase, all those sentence nodes are selected which have matching syntactic and semantic labels;
2. From among those sentence nodes which obey the restriction of part 1, some subset must be chosen whose direct (or indirect) connection pattern is identical with the connection pattern of the corresponding nodes in the criterion phrase.

Consider, as an example, the criterion phrase of Fig. 17 and the sentence of Fig. 16. Clearly, both syntactic and semantic labels of nodes ⑦ and ⓑ, and of nodes ⑨ and ⓐ will match properly. However, there exists a path from node a to node ⓑ in the criterion phrase, while no such path exists from node ⑨ to node ⑦. Therefore, the structure matching procedure will not be successful for the given example. On the other hand, it can be easily verified that the trees of Fig. 18 will, in fact, properly match the criterion phrase of Fig. 17.

Several methods are provided in the system for adjusting the matching process. First, the matching algorithm itself is adaptable, since node and branch labels can or cannot be taken into account, and direct as well as indirect connections can be used. Second, it is possible to provide the criterion phrases with a smaller or larger number of syntactic and semantic labels,

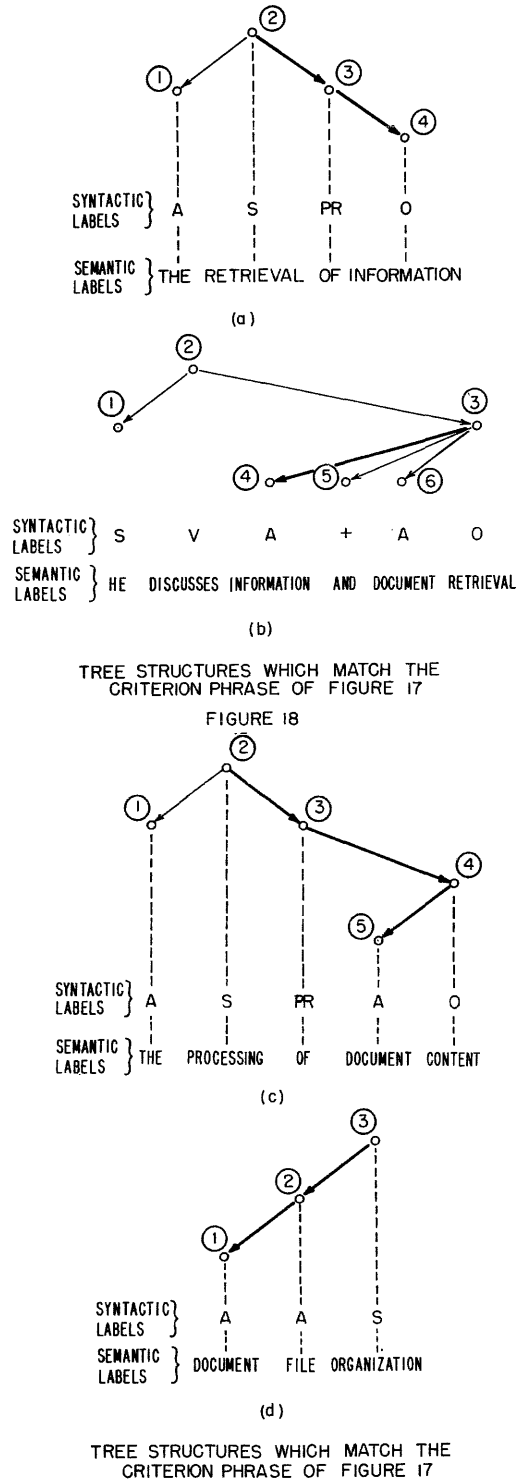


Figure 18. Tree Structures Which Match the Criterion Phrase of Figure 17.

thus restricting or enlarging the possible sentence nodes which are compatible. Finally, the thesaurus which can be used to replace text words by thesaurus categories, as well as the criterion phase dictionary can be enriched to ensure inclusion of a larger variety of possible sentence structures. The system is presently being tested in order to determine the practical effectiveness of these various measures.

REFERENCES

1. M. DÉTANT, Y. LECERF, and A. LEROY, "Travaux Pratiques sur l'Etablissement des Diagrammes," Enseignement Préparatoire aux-Techniques de la Documentation Automatique, *Euratom Report*, Ispra (February 1960).
2. J. C. Gardin and F. LÉVY, "Le Syntol—Syntagmatic Organization Language," *Proceedings of the IFIP Congress—62*, Munich (August 1962).
3. R. BARNES, "Language Problems Posed by Heavily Structured Data," *Communications of the ACM*, Vol. 5, No. 1 (January 1962).
4. G. SALTON, "The Manipulation of Trees in Information Retrieval," *Communications of the ACM*, Vol. 5, No. 2 (February 1962).
5. G. SALTON, "Some Hierarchical Models for Automatic Document Retrieval," *American Documentation*, Vol. 14, No. 3 (July 1963).
6. E. MARDEN and H. R. KOLLER, "A Survey of Computer Programs for Chemical Information Searching," *National Bureau of Standards*, Report No. 6865 (May 1960).
7. H. SHERMAN, "A Quasi-topological Method for the Recognition of Line Patterns," *Information Processing*, Proceedings of the International Conference on Information Processing, Paris (June 1959), pp. 232-238.
8. R. L. GRIMSDALE, F. H. SUMMER, C. J. TUNIS, and T. KILBURN, "A System for the Automatic Recognition of Patterns," *Proc. IEE*, Vol. 106 (1959), pp. 211-221.
9. S. H. UNGER, "GIT—A Heuristic Program for Testing Pairs of Directed Line Graphs for Isomorphism," *ACM National Conference*, Denver (August 1963).
10. E. H. SUSSENGTH, JR., Doctoral Thesis, Harvard University (in preparation).
11. E. H. SUSSENGUTH, JR., "A Graph Theoretic Algorithm for Matching Chemical Structures," *Journal of Chemical Documentation* (to appear).
12. G. SALTON and E. H. SUSSENGUTH, JR., "Automatic Structure - Matching Procedures and Some Typical Retrieval Applications," Report No. ISR-4 from the Harvard Computation Laboratory to the Air Force Cambridge Research Laboratories (August 1963).
13. F. E. HOHN, S. SESHU, and D. D. AUFENKAMP, "The Theory of Nets," *IRE Transactions on Electronic Computers*, Vol. EC-6, No. 3 (September 1957).
14. G. SALTON, "A Flexible Automatic System for the Organization, Storage and Retrieval of Language Data (SMART)," Report ISR-5 to the National Science Foundation, Harvard Computation Laboratory (January 1964).
15. V. E. GIULIANO, "Automatic Message Retrieval by Associative Techniques," *First Congress on the Information System Sciences*, Hot Springs (November 1962).
16. H. E. STILES, "The Association Factor in Information Retrieval," *Journal of the Association for Computing Machinery*, Vol. 8, No. 2 (April 1961).
17. A. LEMMON, "The Criterion Routine," Report No. ISR-5 to the National Science Foundation, Harvard Computation Laboratory (January 1964).

SOME IMPROVEMENTS IN THE TECHNOLOGY OF STRING MERGING AND INTERNAL SORTING

Martin A. Goetz
Applied Data Research, Inc.
Princeton, N. J.

GENERAL

Sort/merge programs for magnetic tape computer systems are of two basic classes:[1]

Digital (or Radix) and
Collation

The digital sort is useful in only a limited number of cases and is not examined in this paper.

The collation sort is the more general type of sort and is composed of two basic sub-programs:

1. The first sub-program internally sorts a group of data. The group of data after being sorted is referred to as a "string" or "initial string." Such a sub-program is referred to as an internal sort.
2. The second sub-program merges two or more strings. It will produce as output longer strings and will eventually produce one string which contains all the data. Such a sub-program is referred to as a "merge," or "string merge."

The input data enters the internal sort sub-program only once and the merge sub-program one or more times. The various sorting systems in use today all try to minimize the execution time of the merge sub-program. This is accomplished by maximizing the way of the merge while at the same time keeping input strings the same size thus resulting in an "effective" power of the merge equal to the way of the merge.* The read-forward Oscillating Technique described in this paper is directed toward this goal.

The Von-Neuman (T/2),[1] Polyphase,[2, 3] and Cascade[4] merge techniques begin after all the data is processed by the internal sort sub-program and the initial strings are placed on tape (Figure 1). The Von-Neuman Technique places the initial strings on only half the available tapes; the Polyphase and Cascade place data on all the unused tapes. Depending on the number of tape units, one of the merging techniques will prove superior over the others. The read-backward Oscillating Technique[5] writes several initial strings, transfers to the merge sub-program, merges the strings and returns to the internal sort—hence the name Oscillating (Figure 2). As the number of tape units used for sorting increase, the effectiveness of the Oscillating Sort increases. Given T input tapes, the effective power of this technique is T–2. This technique, previously considered only for read-backward tape systems, is developed for read-forward only tape systems as described in the following pages.

* For a distinction between "way of the merge" and "effective power of the merge" please see glossary of sort/merge terms.

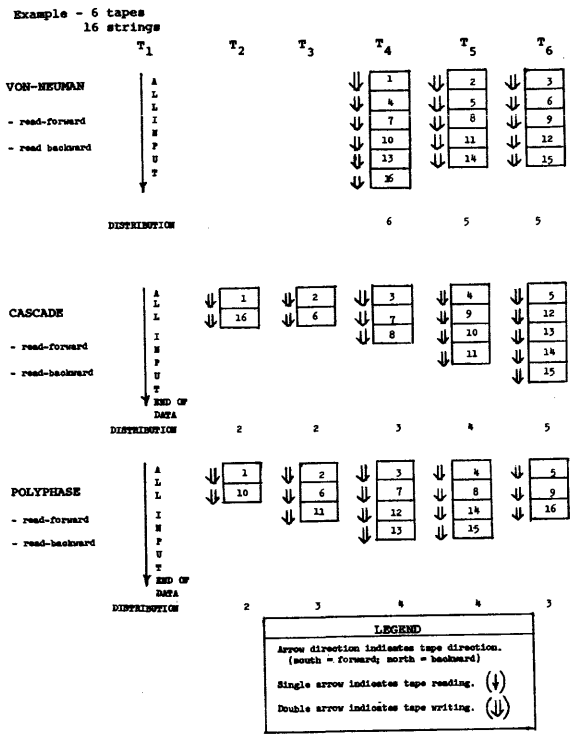


Figure 1.
Distribution of Strings During Internal sort for Von Neuman (N/2), Cascade and Polyphase Merge.

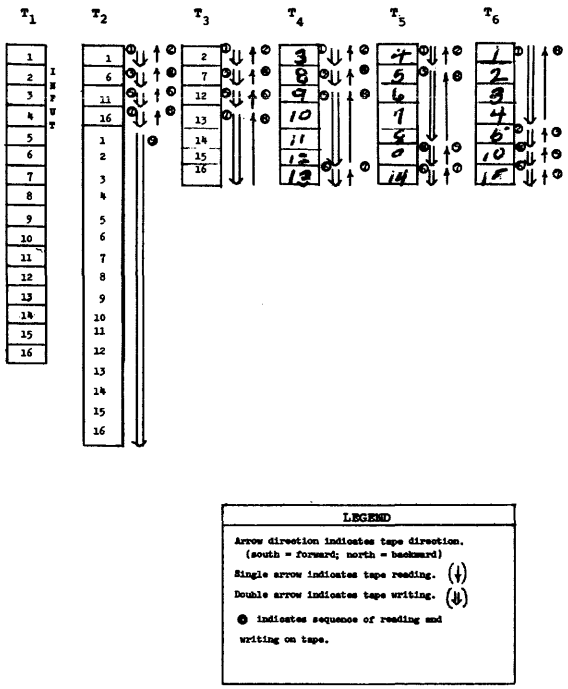


Figure 2.
Tape Motion of Read-Backward Oscillating Merge.

READ-FORWARD OSCILLATING MERGE

The read-forward Oscillating Merge allows the use of a preceding internal sort technique which produces variable size string lengths (e.g., Replacement-Selection[6]). It also allows, of course, techniques which produce fixed-size strings (e.g., successive merging[1]).

This paper presents an example in which the internal sort is a Replacement-Selection. The Replacement-Selection Technique used as an internal sort produces an initial string length almost twice the size of the memory available for sorting. Since the Oscillating Merge Technique must merge initial strings formed by this technique, it is important that we review the nature of the output produced by the Replacement-Selection Technique.

Given a memory for building strings that can hold R records, the expected string length for random data for the first string[7] is 1.73R; for successive strings, 2.0R; without replacement, 1.00R. Given T tape units, the string on the first tape is estimated to be 1.73R; for the last tape unit, 1.00R, for all other tape units, 2.00R. This variability in string length reduces the advantages of the read-backward Oscillating Technique.[8] As will be demonstrated, this variability also reduces the effectiveness of the read-forward Oscillating Technique.

The read-forward Oscillating Technique can also be used with an internal sort that produces initial strings of a fixed size. In this case, the initial string length is 1.0R for all tape units.

THE TECHNIQUE

It is more convenient to describe the technique when the initial string lengths produced by the internal sort are fixed in length (1.0R), It will then be shown how the read-forward Oscillating Merge operates when used with the Replacement-Selection Technique in which variable length strings are produced.

Given N tape units (T_1 to T_N) available as work tapes. (In the example shown in Figure 3, N is equal to 5.)

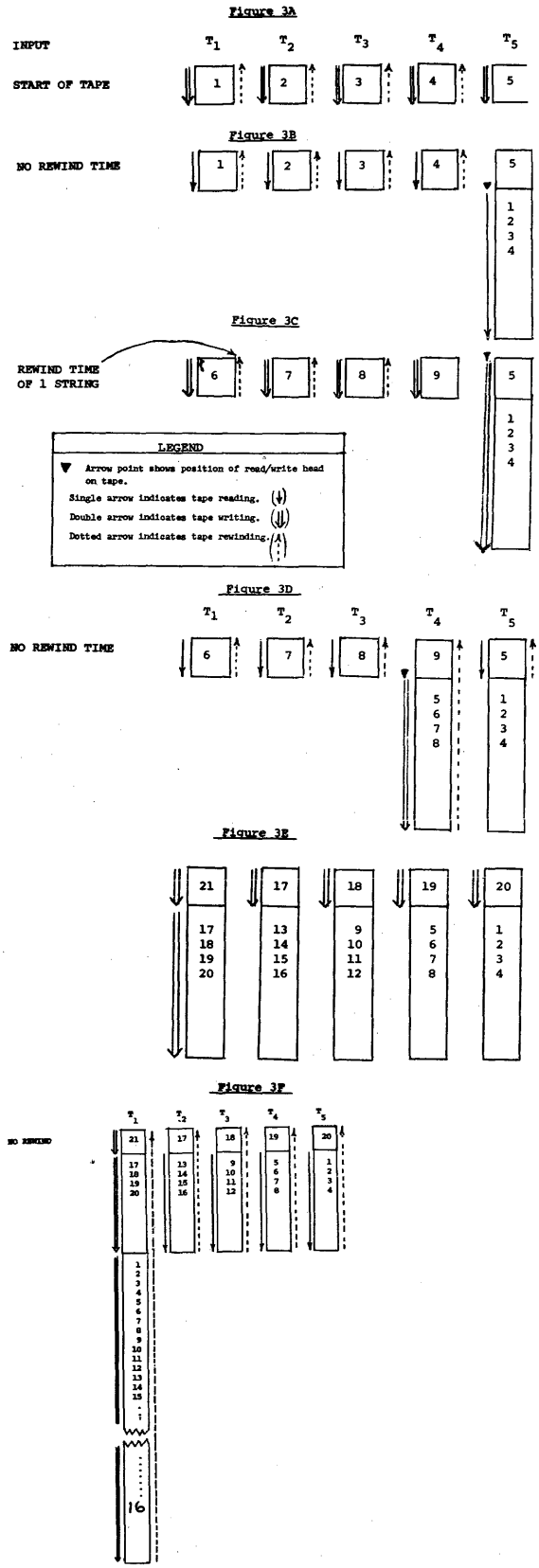


Figure 3.
Tape Motion of Read-Forward Oscillating Merge.

1. Initial strings are written on all tape units (T_1 to T_N). After each string is written on successive tape units, the tape is rewound. The last tape unit is not rewound (Figure 3A).
2. Tape units T_1 to T_{N-1} are merged onto tape units T_N, at which time all tape units are rewound (Figure 3B).
3. As soon as tape unit T_1 is rewound (this tape unit contains only one string of data), a string is written on tape unit T_1 and then it is rewound. Strings are consecutively written on tape units T_2 to T_{N-1}. All tape units except T_{N-1} are immediately rewound (Figure 3C). T_{N-1} is not rewound.
4. A merge of T_1 to T_{N-2} and T_N is performed onto T_{N-1}. Then all tapes are rewound (Figure 3D).
5. The cyclings shown above continue for N cycles at which time all tape units contain a string length of size (N—1) times the initial string length (Figure 3E).
6. The (N—1) size strings are merged, the power being (N—1) as shown in Figure 3F.
7. The pattern shown continues until the data is exhausted, at which time one string will be formed, similar to the read-backward Oscillating Technique.

Note that after 16 strings were merged, the rewind interlock time was equal to rewind time for N—1 strings. This time is a relatively small portion of the rewind time. Note that rewind time is minimized by starting the development of the next string in a group, although it will not be merged until the next cycle (String 21 and Strings 17–20).

When using the Replacement-Selection Technique in conjunction with the read-forward Oscillating Technique, initial strings are constrained not to exceed twice the number of records (R) in memory. After 2.0R records are written on tape, the string is completed. Note that with other merge techniques the minimum string length might be 1.0R as a lower limit and "all" records to be sorted (an entire file) as the upper limit. When using the read-forward Oscillating Technique, the upper limit is set at 2.0R. When 2.0R records can not be formed, dummy records (and blocks)

are substituted to form 2.0R records. Since 2.0R is the expected length, excessive dummys will not be formed. 2.0R may not be an optimum figure. It will, however, closely approximate the optimum as the number of tape units increase.

To the degree that the read-forward selection technique must generate dummy records to produce string lengths of 2.0R, it is less efficient than the read-backward technique. As stated above, however, this reduction in efficiency should be slight under most circumstances.

THE PROBLEM OF WRITING ON TAPE USING THE READ-FORWARD OSCILLATING MERGE

The read-forward Oscillating Merge herein proposed requires that data be written on the front of a tape without destroying information further down the tape which will be subsequently read. Depending on the computer system and the tape units, this may cause problems. For some computer systems, the read head may not be positioned properly due to start-stop time variations, automatic bypassing of unwritable tape (bad spots), effects of the erase head or differences in writing density. For systems with a tape rewrite feature, there would be no problem. For systems which allow tape erase, gaps on the tape can be program generated which will solve the problem. If neither of these features are available "hash" blocks can be inserted to protect information which must be subsequently read. The technique may not be applicable to older systems where there is no programmed error control.

Because of the wide variety of tape systems, this problem is not covered in more detail. It has been investigated, and it can be shown that the additional programming to cope with this problem is trivial.

CONCLUSION

The technque described offers the same potential as the read-backward Oscillating Technique, namely: as the number of tapes increase, this technique will perform the sorting task almost twice as efficiently as the N/2 (Von Neuman) and more efficiently than the Polyphase or Cascade Merge Technique. The Read-Forward Oscillating Technique might be considered even in systems which allow backward reading. This is particularly the case in systems where tape reverse interlock is high.

The foregoing presentation of the read-forward technique should not be construed as a recommendation of this technique to the exclusion of other methods. The selection of proper sorting techniques is a complex problem dealt with in detail in other papers [5, 8, 9, 10] and which is not completely formalized at this time.

VARIABLE LENGTH RECORD SORTING USING THE REPLACEMENT-SELECTION TECHNIQUE

INTRODUCTION

Another area where efficiencies in sorting can be attained, is in the internal sort. Since the number of merging passes is based on the number of internal strings, it is desirable to minimize the number of strings formed by the internal sort. The number of strings are minimized when the amount of data sorted at one time (the length of the string) is maximized.

The sorting techniques in use today limit the string length for variable size records to the number of records that can be stored in memory. This section describes a sorting technique that permits an initial string to be formed that contains approximately twice as much data as can fit into the working storage available in memory.[7]

GENERAL

The variable-record length internal sort segment uses a modified version of the Replacement-Selection Technique.[6] This technique has previously been applied to the sorting of fixed-size records or to a variable-size record converted into a fixed format.

The proposed technique temporarily "disjoins" a variable-size record into one or more fixed-size "pieces" (referred to as segments) and at selection time combines the separate segments of the record. *No expansion of the record occurs.*

The selection of an "optimum" fixed-size segment storage area is either determined by the user (based on his knowledge of the data) or assigned by the computer program. The optimum size is one which will produce the longest string on tape without causing average size records to be broken up into a large number of segments. Long strings are produced by selecting a fixed storage segment size into which the records will fit without the need for excessive "fill" when the last "segment" of the record is moved into the fixed segment area. The selected segment size should be such that all keys for each record appear in the first disjointed segment when subdivided.

A brief review of the selection logic for conventional fixed size Replacement-Selection internal sorting follows:

Core memory is divided into 4 parts.

(1) Instructions
(2) Input Areas
(3) Output Areas
(4) A string building area (work storage area)

The string building area is further subdivided into pockets (or slots) equal to a fixed size plus about 12 characters to hold information used during the selection process. The selection process is similar to a tournament match in which the selected record at each level is referred to as a "winner." Assume as an example, the string building area can contain 12 records. The logic for selection is as follows:

1. A string building area is filled with records (R) from the input area. (See Figure 4.)
2. The first record in the string building area is compared against the second, the third against the 4th, etc. In this manner a set of first round winners is selected. The addresses of the winners are stored.
3. In a similar way succeeding rounds of winners are selected until one final winner is selected. Addresses of the winners of each round are recorded. This process is called initialization of the tree and requires R-1 comparisons. (See Figure 5.)

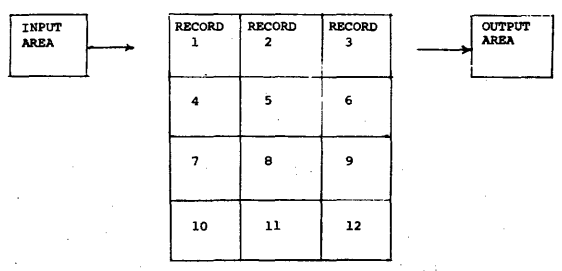


Figure 4.
Example of Records in String Building Area.

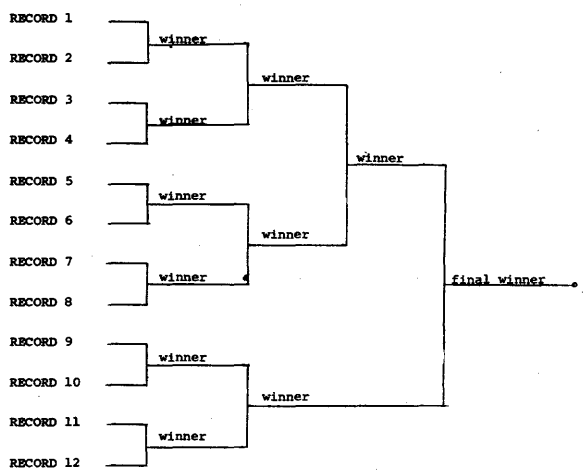


Figure 5.
Logical Relationship of Records in Tournament.

4. The final winner is moved to the output and its position in the string building area is replaced with a new record from the input area.
5. If the new input record could be put out in sequence with respect to the last record that was put out, it can participate in the current tournament. Otherwise, it may not participate in the current tournament.
6. Each subsequent record selected requires Log_2R comparisons. This selection process is referred to as a "scan."
7. When no records can participate, the current tournament is over and a new string must be formed.

USING THE REPLACEMENT-SELECTION TECHNIQUE FOR VARIABLE LENGTH RECORDS

A record in the input area is divided so that one or more segments of it may be moved into

slots in the string building area. The first such segment, containing the key, is called the header; all other segments are trailers. Each segment of a record, except the last, contains as part of its indicative information, the address of the segment of the record which is its immediate successor.

When the string building area is completely filled it is "initialized." Only headers participate in this initialization and only headers are winners in each "round" of the tournament. Initialization is concluded when a final winner has been determined.

At this point, the winner is moved to the output area. Immediately, its place in the string building area is filled from the input area. Whenever a "segment" is moved from the string-building area, that slot is immediately filled. This is true irrespective of whether a header or trailer segment was moved out (Figure 6). As segments are moved out they are rejoined in the output area into a variable length record. Any "fill" in the last segment is deleted at this time.

After a string building area slot has been refilled, a "scan" takes place if one is necessary. It is necessary to "scan" if either a header was moved out of the string building area or if a header was moved into the string building area. In other words, the only time a scan is not performed is when a trailer is replaced by a trailer.

After the scan, the next "segment" is moved into the output area according to the following rules:

1. If the last segment moved to the output contained a reference to a successor (which must be a trailer), the successor is moved. This accomplishes the result of assembling the records which were segmented initially.
2. If the last segment moved was without a successor, the current final winner of the tournament (which must be a header), is moved to the output.

FIGURE 6

RECORD SEGMENTS IN STRING BUILDING AREA

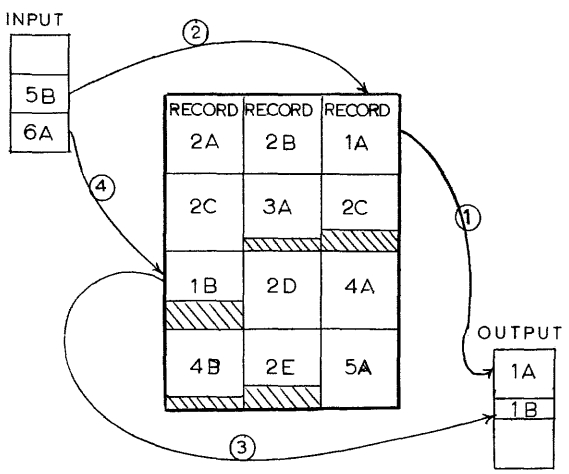


Figure 6.
Record Segments in String Building Area.

CONCLUSION

The logic for processing variable-length records using the Replacement-Selection Technique requires an additional address in the tournament tree which is used to chain between segments of a variable-length record. Additional logic is required to process variable length records. Records are segmented into header and trailers. After the selection process, they are recombined into a variable length record. The logic of the Replacement-Selection Technique requires modifications as described. As in the Replacement-Selection Technique for fixed size records, data is moved only twice. The overall logic of the Replacement-Selection Technique is retained and all its advantages are exploited.[6]

ACKNOWLEDGEMENT

The author wishes to express his appreciation to Mr. Warren F. Spalding of Applied Data Research, Inc. for his assistance in the organization and editing of this paper.

GLOSSARY OF SORTING AND MERGING TERMS USED WITHIN THIS PAPER*

Backward read

A feature available on some magnetic tape system whereby the magnetic tape units can transfer data to computer storage while mov-

ing in a reverse direction. Normally used, if available, during the external sort phases to reduce rewind time.

Balanced sorting

See: (T/2)-way merging.

Cascade merging

A technique used in a sort program to merge strings of sequenced data. Given T work tapes, merging is performed at T-1 on part of the data, T-2 on parts of the data, and so on. Strings of sequenced data are distributed in a Fibonacci Series on the work tapes preceding each merge. The effective power of the merge varies between T-1 and T-2 but in all cases is less than the power of the Polyphase Merge. CF: effective power of the merge.

Collating

Sequencing a group of records by comparing the key of one record with another record until equality, greater than, or less than is determined.

Collating sequence

The sorting sequence; a description of the sort key for a file of records.

Collating sorting

A sort which uses a technique of continuous merging of data until one sequence is developed.

Computer limited

See: process limited.

Digital sorting

A sort which uses a technique similar to sorting on tabulation machines (e.g., IBM Sorter). The elapsed time is directly proportional to the number of characters in the sequencing key and the volume of data. Also "radix sort."

Effective power of the merge

Equal to $S\frac{1}{N}$, where S is the number of input strings and N is the average number of times each element of data is read.

Fibonacci series

A series where the current number is equal to the sum of the two preceding numbers: i.e., 1, 2, 3, 5, 8, and so on. Some sort programs distribute strings of data onto work tapes so that the number of strings on successive tapes form a Fibonacci series.

Fixed size records

Denumerable file elements each of which has the same number of words, characters, bits, fields, etc. Cf: variable-length records.

Generalized sort

A sort program which will accept the introduction of parameters at run time and which does not generate a program.

Generated sort

A production program which was produced by a sort generator.

Input tape(s)

Tape(s) containing a file in arbitrary sesequence to be introduced into a sort/merge program.

Insertion method

See: Sifting.

Item

See: Record.

Key

Also, sequencing key; criteria; sequencing criteria. The fields in a record which determine, or are used as a basis for determining, the sequence of records in a file.

Magnetic tape sorting

A sort program that utilizes magnetic tapes for auxiliary storage during a sort.

Major key

The most significant key in a record.

Merge

A program that performs merging.

Merging

The forming of a single file of sequenced records from two or more files of sequenced records.

Multifile sorting

The automatic sequencing of more than one file, based upon separate parameters for each file, without operator intervention.

* Source: A Glossary of Sorting and Merging Terms —Communications of ACM, May 1963.

Multipass sort

A sort program which is designed to sort more data than can be contained within the internal memory of a central computer. Intermediate storage, such as disc, tape, drum, etc. is required.

Optimum merging patterns

The determination of the sequence in which specific sorted tapes in a file should be processed so as to minimize the total number of merge passes required to create a single file of sequenced records.

Order of the merge

The number of input files to a merge program. Also: power of the merge.

Oscillating merge

A technique used in a sort program to merge strings of sequenced data. For tape systems that permit backward reading, the effective power of the merge is equal to T-2.

Output tape(s)

Tapes containing a file in specified sequence as a result of a specific sort/merge process.

Pass

The processing of each file record once for the purpose of reducing the number of strings of sequenced records and increasing the number of sequenced records per string.

Phase

An arbitrary segmentation of a sort program. Many sorts are segmented into three phases: initialization phase, internal phase, merge phase.

Polyphase merging

A technique used in a sort program to merge strings of sequenced data. Given T work tapes, merging is performed at the power of T-1. The effective power of the merge varies between T-1 and T-2 depending on the amount of input data and the number of strings.

Power of the merge

Also: way of the merge; order of the merge, the number of inputs to a merge program. Cf: effective power of the merge.

Process limited

Also: computer limited. A sort program in which the execution time of the internal instructions determines the elapsed time required to sort. Cf: tape limited.

Radix-sort

See: Digital sorting.

Record

The basic element of a file such that the sorting of file constitutes the re-ordering of file records; also referred to as "item."

Replacement-selection technique

A technique used in the internal portion of a sort program. The results of the comparisons between groups of records are stored for later use. A selected record is placed on the output tape and a new record replaces the selected record. Given N records, a record is selected with $1 + \lg_2 N$ tests; the expected string length for random data is 2N records.

Rewind time

Elapsed time consumed by a sort/merge program for restoring intermediate and final tape files to original position.

Scratch tape(s)

See: Work tapes.

Sequence break

That point in a file between the end of one string and start of another.

Sequencing criteria

See: Key.

Sequencing key

See: Key.

Sifting

A method of internal sorting where records are moved to permit the insertion of records; also called "insertion method."

Sort

The copying of a file of records into a corresponding file in a specified sequence.

Sort, external

The second phase of a multipass sort program, wherein strings of data are continually merged until one string of sequenced data is formed. Cf: string merge.

Sort, internal

The sequencing of two or more records within the central computer memory; the first phase of a multipass sort program.

Sort generator

A program which generates a sort program for production running.

String

A group of sequenced records, normally stored in auxiliary computer storage; i.e., disc, tape or drum.

String merge

Program that performs merging.

String merging

The forming of a single string from two or more strings of sequenced records.

(T/2)-way merging

A technique used in a sort program to merge strings of sequenced data. The power of the merge is equal to T/2.

Tape limited

Also: I/O limited. A sort program in which the effective transfer rate of tape units determines the elapsed time required to sort. Cf: process limited.

Tennis match sorting

See: Replacement-Selection Technique.

Tournament sorting

See: Replacement-Selection Technique.

Variable-length records

Denumerable file elements for which the number of words, characters, bits, fields, etc. is not constant. Cf: fixed-size records.

Von Neuman sort

See: (T/2)-way merging.

Way of the merge

See: power of the merge.

Work tape(s)

Also: scratch tapes. Tape(s) used to store intermediate pass data during a sort program.

Xmas tree sorting

See: Replacement-Selection Technique.

REFERENCES

1. FRIEND, E. H., "Sorting on Electronic Computer Systems," Journal ACM, July 1956.
2. GILSTAD, R. L., "Polyphase Merge Sorting—An Advanced Technique." Proceedings of the Eastern Joint Computer Conference, December 1960.
3. GILSTAD, R. L., "Read Backward Polyphase Sorting." Communication ACM, May 1963.
4. BETZ, B. K., and CARTER, W. C., "New Merge Sorting Techniques." Paper 14, Reprints of Summaries of Papers, 14th National Meeting ACM, 1959.
5. SOBEL, S., "Oscillating Sort—A New Merge Sorting Technique." Journal ACM, July 1962.
6. GOETZ, M. A., "Internal and Tape Sorting Using the Replacement-Selection Technique." Communications ACM, May 1963.
7. KNUTH, D. E., "Length of Strings for a Merge Sort." Communications ACM, November 1963.
8. GOETZ, M. A., "Comparison between Polyphase and Oscillating Sort Techniques." Communications ACM, May 1963.
9. KNUTH, D. E., and GOETZ, M. A., Letter to the Editor, Communications ACM, October 1963.
10. HALL, M. A., "A Method of Comparing the Time Requirements of Sorting Methods." Communications ACM, May 1963.

CONCEPTUAL MODELS FOR DETERMINING INFORMATION REQUIREMENTS

James C. Miller
Arthur D. Little, Inc.
Cambridge, Massachusetts

INTRODUCTION

For years, we who are interested in data processing have had a vague notion that one of the problems facing managers today is the lack of information. "If only I had known . . ." is a familiar phrase to all of us. Most of us would like to cause that phrase to become unfamiliar. Unfortunately, very few people, if any, have been able to state very explicitly how we should go about filling the information void. Progress in developing a methodology for designing management information systems has been slow.

So many people have written and said so much about management information systems that I would like to be sure that all of us have a similar notion in mind. Therefore, I would like to briefly define a management information system as—a collection of procedures, equipment, and persons associated together for the purpose of providing managers, who have the authority to make decisions that commit the firm or its resources, with descriptions of the elements relevant to the performance of their function. In other words, a management information system is a means of providing to the people who "need" it, information to guide them in the conduct of the business. An ideal management information system, then, would do at least these things:

1. Provide each level and position of management with all the information that can be used in the conduct of each manager's job.
2. Filter the information so that each level and position of management actually receives only the information it can and must act on.
3. Provide information to the manager only when action is possible and appropriate.
4. Provide any form of analysis, data, or information whenever it is requested.
5. Always provide information that is up to date.
6. Provide information in a form that is easily understood and digested by the manager.

Whether or not you can fully agree with this as a description of a managerial utopia, I hope that this will at least provide a frame of reference that will help you to understand the use of conceptual models.

PROGRESS IN MANAGEMENT INFORMATION SYSTEMS

Using this ideal as a standard toward which our efforts are directed, we must recognize that no one has an ideal management information system. Just because we have not yet reached our ideal, however, we should not be discouraged from attempting to make progress. It is not easy to produce a monumental improvement in the science of management. Nevertheless, considerable progress has been made. In fact, progress has been striking in the fields of hardware, software, mathematical techniques, and

the integration of procedural systems. Computers are available that can produce information at fantastic rates. High-speed printers can generate reports fast enough to inundate an entire committee. Teleprocessing has been developed to the point where up-to-date information can be maintained in a central data file. All of these various types of hardware can and will be improved, and all of them will probably be made less expensive or more efficient. However, I believe that management is not utilizing the capability that is already available.

Software has been developed so that special requests or changes in procedures need not totally upset a system. I am not beating the drums for FACT, COBOL, or any other programming language; I am sure that all of these will be improved further. However, I believe that even software is available today to assist management more effectively than managers realize.

Mathematical techniques have been developed for many things. We can optimize inventories, we can model markets, and we can predict the outcome of an election when only a small percentage of the returns have been reported. Even so, this resource of mathematical techniques is neither well-understood nor fully utilized by managers.

Many systems and procedures people have made substantial progress in developing integrated data processing systems. These systems chew up customers' orders at one end and spew out bills of lading, invoices, and production orders from the other end. However, their primary focus of attention has been on the routine operating documents of the business. They have made a substantial contribution in that they have permitted many managerial tasks to be directed by management instead of being subject to the individual judgments of many operating people. In inventory control, for instance, stock clerks are no longer responsible for inventory levels; management has the key to the inventory control system and can adjust its mathematical judgment to management's will.

The point of all this is that in spite of tremendous progress on a countless number of fronts, managers are not truly helped. Generally speaking, managers are working with the same variety of reports that they had several years ago. We believe that a major area of systems design has not been given sufficient attention. The problem of developing and defining the proper content of an information system has been slighted in the general work of systems analysis and design. One of the reasons for its having been slighted is that it typically falls in a no man's land between the technician and the manager. The technician typically says to the manager, "All you need to do is tell me what you want, and I have the wherewithal to supply it." Whereupon the manager, out of desperation, lack of foresight, or overconfidence, usually supplies the pat answer: "Just what I am getting now only quicker and more current." Some managers will honestly say: "I don't know, but you're the systems expert; can't you tell me?"

This no man's land has created considerable difficulty, and very few people have risen to the challenge to try to do something about it.

Another reason for little having been done about the ability to define the content of an information system is that content is extremely difficult to work with.

Let me define content as the message or information that is contained in a communication or a record. Every report, analysis, or document has some meaning (or message) that transcends the actual format of the document or report. When we are concerned with defining the content of a management information system, our concern is to determine the subject matter of the messages that managers should receive. When I talk about content, I am referring to the subject matter of reports and documents, regardless of how the data is displayed or arrayed.

Information content is difficult to determine for managers. It is almost impossible to separate the content of managerial information from the field of organization theory. Organization theory and practice is thin ice; it is a subject that is emotionally charged for any manager in a real-life situation. It is also a subject on which highly qualified, reasonable men can be expected to disagree. This is an extremely difficult area for anyone to work in, and particularly for a person who is scientifically and analytically inclined. You cannot determine the

information that a manager needs without considering his responsibility and authority. You must concern yourself with what he is, in theory, held responsible for, and how he discharges his responsibility and delegates his authority.

Being concerned with the content of an information system forces us to be concerned with "how to manage well." We must concern ourselves with: How does a manager operate? How does he reach his decisions? How does he make his decisions effective? How does he manage?" And perhaps an even more appropriate question is: "How should he manage?" No one has an adequate description or an adequate set of principles to tell us how to manage well. At best, there are a thousand platitudes that are collectively exhaustive and mutually contradictory. As if the lack of knowledge and understanding of the subject (from any analytical point of view) were not bad enough, this is also a subject that managers have difficulty discussing rationally and on which respected authorities disagree.

In spite of the difficulty, we believe that this is one area in which substantial progress will be made in the next decade. If nothing else, managers and systems men will be forced to it by the availability of hardware, etc., and the fear that some competitor may do it.

We believe that much of this difficulty is mental, and that we can attempt to develop a method for determining information requirements by trying to make the job mentally easier. Therefore, what we have to suggest is not so earth-shaking, but we believe it is a sound approach than can enable a poor, mere, mortal mind to somehow get around the subject of management and get into the business of defining information requirements. The approach that we have to suggest might be described as a research approach. It is an approach that should enable analytically-inclined people to develop a definition of their company's information needs. If they want to, they can complete the entire job in an ivory tower, but the job will be done better if they have frequent reference to the regular, operating facts. The analysts should not lose touch with reality, but in fact, we have used this technique when there has been no reality to get in touch with. We have developed information requirements for nonexistent firms, and we believe that the results were extremely satisfactory.

Of course, an approach that is frequently productive, but is not the research approach, is that of expertise. Most information systems and most informational improvements that are made today are made on the basis of expertise. Someone writes an article in a professional journal or a scholastic business review describing the types of reports that they use. Managers read these reviews, think that they are wonderful and try to apply them to their business. The formats of the reports are face-lifted, and the manager tries to use them. Sometimes he has great success, and sometimes no success at all.

There are many varieties of models, and they can be used in many different ways. Operations research people are accustomed to applying mathematical models to business problems. Some chemists and biochemists use physical models of what they imagine the structure of atoms to be. For now, we are interested in dealing with a conceptual model. It is a model that deals with words and imagery to enable us to focus our attention and communicate our impression about the operations and the management of an enterprise.

MODELS OF OPERATIONS

The first step in developing a conceptual model of an enterprise is to attempt to state the key operations that the enterprise must accomplish in order to continue to function. We might describe an operation as a "gross unit of work specialization that is essential to the functioning of the enterprise." The easiest way to determine the appropriate operations for a concern is simply to begin to list all operations. Once the initial top-of-the-head list has been compiled, it should be juggled, combined, expanded and organized until it consists of a number—probably between 10 and 20—of operations of approximately equal importance.

As an example, I have chosen a wholesaling business. We might think of this as being a typical wholesaling business rather than any one specific wholesaler. Figure 1 is an initial list of potential operations for a hardware, drug, or appliance wholesaler. Many of these

captions appear to be steps in a procedure, and they should. Most business operations follow a routine. On the other hand, very few of the captions suggest a department or organizational unit of a wholesaler. We believe that the major concern of management is the basic operations of the business. If management loses sight of this and becomes preoccupied with people, the business can become a very nice place to work, but an extremely disorganized mass of human relations. We want to concentrate on the basic operations.

STOCK CHECKING	MAINTAIN PRODUCT LINE
ORDERING	TELEPHONE SELLING
SCHEDULING	SALESMAN SELLING
RECEIVING	ADVERTISE AND PROMOTE
STOCKING - BULK	WRITE ORDERS
HOLDING	PICKING
STOCKING - SHELF	CHECKING
OK RETURNS	DELIVERY
PICKUP RETURNS	INVOICE
CREDIT RETURNS	BILL
STORE RETURNS	COLLECT
SHIP RETURNS TO MFR.	MAINTAIN ACCOUNTS
RESTOCK RETURNS	

Figure 1. Potential Wholesaling Operations.

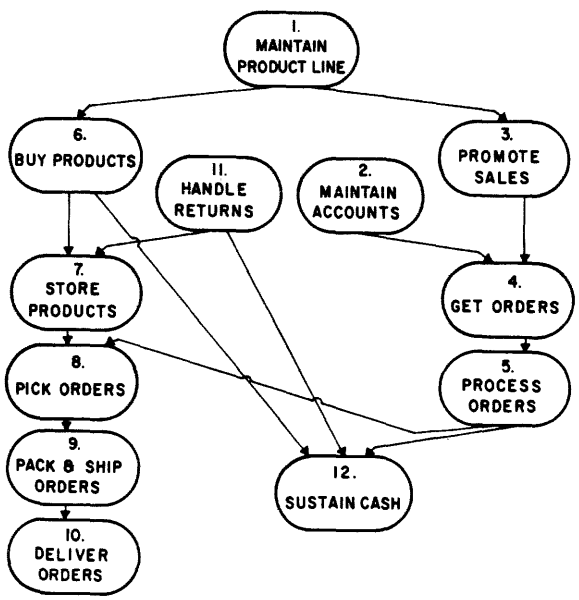


Figure 2. Operations of a "Typical" Wholesaler.

Figure 2 shows the operations that I finally selected as being the important ones for a typical wholesaler. They are arranged in a flowchart format. The reasons for this will be explained later. Each block represents a job, task, function, mission, or as we have chosen to call them, operation of the firm. Take any one of these blocks out of the diagram and the business either ceases to exist, or is changed drastically in its methods of operation.

In effect, this diagram of the operations of a wholesaler is a macroscopic view of the business. Businessmen are already acquainted with other models of their business. They are accustomed to an organization chart and they are familiar with accounting statements. All of these are macroscopic views of the business.

The view of the firm that is illustrated in Figure 2 is specifically designed to focus our attention on the important things that must be managed. All of these things must be under the control and the close scrutiny of management. Each of these operations can be accomplished in a variety of ways. Each of these operations is a positive reason for spending money, not just an unavoidable expense—we have not included the payment of taxes or the negotiation with labor unions, nor, in fact, have we included the information system itself.

The next step is to try to provide meaning to each of the names that has been put in a box in our flowchart. Because these names mean different things to different people, it may appear that we have omitted a significant operation, or we might have difficulty agreeing that a certain activity is included in one operation rather than another. Before we go any further, we should provide a more careful delineation of each operation. This delineation should take two forms. The first form is a simple statement of the input and the output for each operation. Inputs and outputs are, in effect, the fences between the operations. They serve not only to help define the operations themselves, but also

to make us certain that we have not omitted some significant activities between the operations. Figure 3 shows the wholesaler's operations with their inputs and outputs. Notice that the only original input is product ideas and the only final outputs are: goods with customers, payment, cash, and collections. Having all the operations and all the inputs and outputs together in a single flowchart is a help, but it is still just a body of names which do not yet have sufficient meaning to enable us to probe the process of management.

The second form of delineation for operations is a description of the suboperations that are contained within the major operation. Figure 4 is a statement of the suboperations that are required to transform the demand for specific products, customers ready to buy, and preference for our services into orders. These lists of suboperations can easily appear to be a description of a general operating procedure. They are likely to include some of the things that we jotted down as potential operations in the very beginning. (See Figure 1.) If we were dealing with a specific firm, the description would be more detailed.

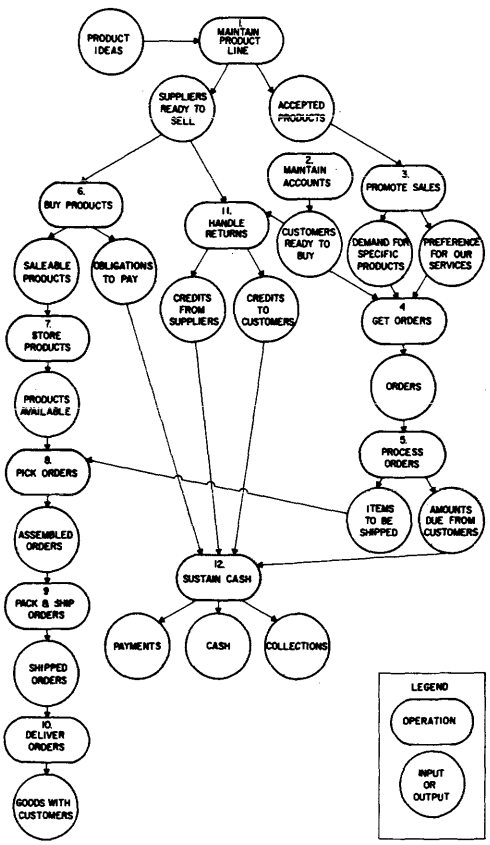


Figure 3. Wholesale Operations with Inputs and Outputs.

OPERATION DESCRIPTION

4. GET ORDERS

ORDERS SHOULD BE SECURED FROM RECOGNIZED CUSTOMERS. THESE CUSTOMERS SHOULD BE GIVEN ANY APPROPRIATE ORDERING AIDS SUCH AS WANT BOOKS, ORDER BLANKS, PREPAID ENVELOPES, ETC. FOR FOR CUSTOMERS IN REMOTE CITIES, LEASED TELEPHONE LINES WITH LOCAL NUMBERS MAY BE PROVIDED TO ENABLE CUSTOMERS TO PLACE ORDERS WITHOUT PAYING LONG-DISTANCE TOLL RATES, OR CUSTOMERS MAY BE ENCOURAGED TO CALL COLLECT.

THE CUSTOMER'S INVENTORY MAY BE REVIEWED FOR HIM BY A REPRESENTATIVE OF THE WHOLESALER (POSSIBLY THE SALESMAN). THE WHOLESALER MAY MAINTAIN A PERPETUAL INVENTORY RECORD FOR THE WHOLESALER. AUTOMATIC DECISION RULES MIGHT BE PROVIDED AND REVIEWED BY THE WHOLESALER.

A TELEPHONE CALLING SERVICE MAY BE ESTABLISHED TO SOLICIT ORDERS FROM CUSTOMERS. THIS INCLUDES: (1) ESTABLISHING AND SUPERVISING A TELEPHONE SALES STAFF; (2) SELECTING THE CUSTOMERS TO BE CALLED AND ESTABLISHING A SCHEDULE FOR THE CALLS THAT IS GEARED WITH OTHER OPERATIONS OF THE WHOLESALER; (3) CALLING CUSTOMERS; (4) SUGGESTING ITEMS AND QUANTITIES TO BE ORDERED; (5) QUOTING PRICES; AND (6) PREPARING ORDER DOCUMENT.

ORDERS MAY BE SOLICITED DIRECTLY BY SALESMEN WHO PREPARE THE ORDER DOCUMENT, GET THE CUSTOMER'S APPROVAL, AND SEND IT TO THE WAREHOUSE.

"TURNOVER" ORDERS MAY BE SECURED BY ENCOURAGING CUSTOMERS TO SPECIFY OUR NAME TO MANUFACTURER'S SALESMEN. ALSO WORK WITH MANUFACTURER'S SALESMEN TO INCREASE THE PROPORTION OF THEIR ORDERS THAT ARE TURNED OVER TO US.

Figure 4. Description of an Operation.

This (Figures 3 and 4) completes a conceptual model of the firm. Some of my associates and I have gone through this exercise for many firms and some command and control situations. We have found in every case that when two, three or four people sit down to prepare this conceptual model of an activity, they can, by constant negotiation give and take, agree upon a set of operations and definitions of operations. In short, a number of people with different backgrounds can follow this procedure to produce a single, well-defined, comprehensive view of the activities of a company.

MODEL OF MANAGEMENT ACTIONS

Now that we have a conceptual model of what the firm, as a whole, does, we would like to move on to a conceptual model of the functions

of management. We have an adequate statement of what the company does, but we must now decide how management manipulates the things that the company does in order to make it successful or unsuccessful. The basic question can be simply stated as: "How are the operations managed?"

Before getting too deeply involved in the conceptual model of management actions and their results, let's spend a little time poking into the lore of management. Many books have been written, from Frederick Taylor until the present time, about how managers can and should operate. The business reviews of our leading universities constantly publish articles about how to manage. We commend these sources to your attention. From our study of these sources, we have generalized and concluded that management must evaluate, organize, select, decide, train, and motivate. We can also recognize that management has at its disposal a number of resources. Resources can be summarized into the "four M's"—money, machines, manpower, and materials. Somehow, citing these names (evaluate, etc.) for the things that managers do, and citing the names for the resources that management manages seems to be helpful, but it cannot be the final conclusion. These names and labels are not sufficiently specific to help us decide what information management needs in order to manage effectively.

In addition to the lore of management, we can logically consider the things that a manager does in a typical work day. Those who are managers, and those who are familiar with the general operation of managers, can recognize that many of the things that a manager does do not have long-range significance for the company as a whole. For instance, a large part of the working day for a manager is spent communicating with those about him. This communicating is, of itself, not truly significant. It is an unavoidable expense. The decisions that may result from those communications, or the evaluations that can be made as a result of them, are significant, but the communications themselves are not. Similarly, a good manager spends a large part of his time studying and reading reports. In reality, this is simply another form of communication. The amount of time that a manager spends actually making policy and making key decisions is a relatively small proportion of his total time. However, we believe that these are the significant things that a manager does that we would like to assist. With a good information system we might be able to reduce the amount of time that a manager must spend communicating and reading reports. But more significantly, we would like to assist him in making wise decisions about the truly important facets of the operations that he is managing.

One way to help cut out some of the chaff is to recognize that we are attempting to delineate those important managerial actions that are taken by management as a whole. We are not trying to find out the specific actions that any one manager takes. If we were, we would miss the significant managerial actions taken by committees. In many cases, a managerial action is taken at different levels of the organization. At each level, the manager has a different set of limitations within which he makes his decision. If we try to cope with all of these variations at once, we will be swamped. Our immediate concern is for the information that is required by management as a whole. We would like to postpone until much later the actual job of deciding exactly who should receive which information.

After many trials and errors, and considerable study, we have concluded that the most significant managerial actions can usually be stated as "selecting a course of action," "adjusting a rate of expenditure (or level of effort)," or "allocating resources." In a sense, allocation of resources is simply a combination or special case of selecting and adjusting.

To illustrate, a manager selects a course of action when he decides to use a particular channel of distribution, or decides to acquire a particular piece of production machinery. In general, these are discrete choices; the manager must select one or more out of a number of alternatives.

On the other hand, a number of elements can be adjusted over a continuous spectrum, such as prices or market area. These things can be adjusted up or down over a broad range.

Allocation consists of assigning particular resources to particular activities. The word

"allocation" suggests that the amount of the resource is limited in some way. The grandest sort of allocation is the allocation of money to each of the operations conducted by the firm. The general management of the firm must constantly reallocate its money among such activities as selling, delivery, and inventory.

Keeping in mind these general sorts of managerial actions, we can return to the conceptual model. We must examine each operation to determine the significant managerial actions that govern the quality of performance of each operation. If management acts wisely, the operations will be performed well. A management information system can not supply good judgment, but it can supply a sound base of facts to which managerial judgment can be applied. Figure 5 shows the managerial actions for the operation "Get Orders." Discovering these managerial actions for any operation is one of the most creative and imaginative steps in the process of constructing a conceptual model of a firm. It takes time and it takes stargazing. It is an iterative process that can be improved each time it is reviewed. It is also an extremely educational process. If the managers themselves can participate in the process, they can probably profit by it.

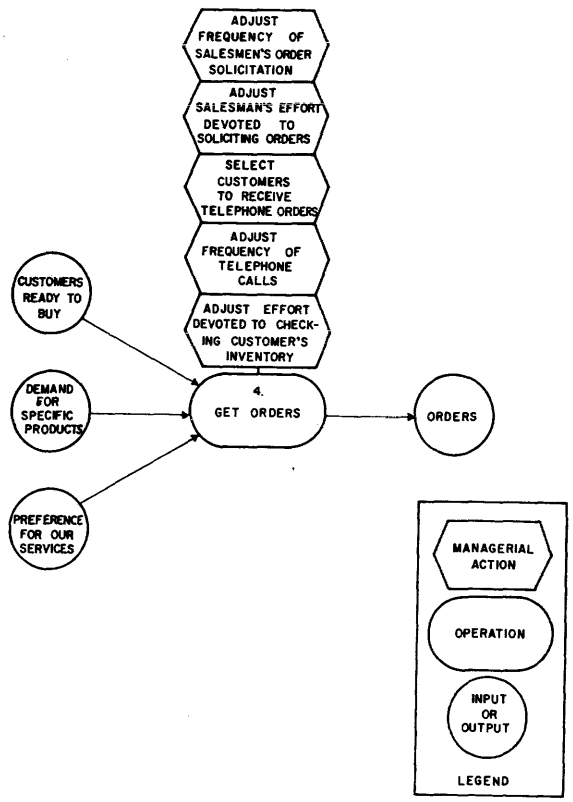


Figure 5. Managerial Actions for an Operation.

There are a few sources that we can look to for assistance in pointing out the key managerial actions. One of these is the detailed description of the operation. (See Figure 4.) We can review that description, looking for instances in which a manager must select from a number of alternatives, or for key decisions that are built into the regular conduct of the operation. We can also consider the resources that are required to perform the operation. It might pay to construct a list of the resources that are used in each operation. Resources might be:

Particular skills
Manual labor
Existing facilities (physical capabilities)
Known suppliers
Existing public (customer) image
Existing products

Figure 6 lists the major resources that are used to get orders. Some of these resources are subject to quantity manipulation. However, the rough proportions of the various resources are dictated by the nature of the operation itself. For instance, for a wholesaler, the selling activity cannot effectively use a large fixed capital investment; almost the sole resource for selling is the highly skilled ability of a salesman in personal contact with the customer.

In considering the resources required to perform an operation, there is a potential trap. That trap consists of considering money as a resource. No one can deny that money is a resource, but it is the one ultimate resource. Given sufficient time, it can be transformed into

- SALESMAN'S TIME
- TELEPHONE SALES CLERK'S TIME
- FAVOR OF MANUFACTURERS
- TELEPHONE FACILITIES

Figure 6. Resources Used in Getting Orders.

any of the other resources. Therefore, in determining the resources that are used in the performance of an operation, we should exclude money from our consideration. Otherwise, we run the risk of doing a superficial job.

Another potential source of help in discovering managerial actions are job descriptions, organization charts, financial statements, and interviews with managers. All of these aids should be used liberally.

Next we would like to consider the results of each managerial action. Usually, at least one result of every managerial action is obvious from the statement of the action itself. If the action selects or adjusts, one result of the action is a commitment to a course of action or a change in the level of something. However, we are interested not only in the direct effects of the action itself, but also in the ancillary effects. Almost every managerial action involves more than one result. Many managerial actions imply a trade-off between two potential results. Some managerial actions simply have more than one effect. Figure 7 shows two managerial actions and their results. In total, when taken for all managerial actions, these form a conceptual model of a management of the firm.

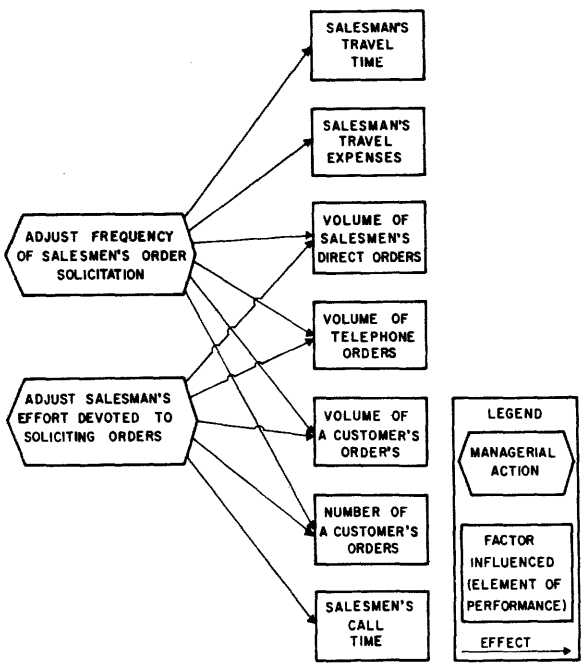


Figure 7. Action-Result Models.

We slipped into using the word "result" rather quickly. We might better call them elements of performance, or parameters of performance. These are the factors or elements in the business that are influenced by the managerial actions. It is important to think of the relationship between the action and its results as an influence. If you try to think of it as too direct a cause-effect relationship, you are likely to get bogged down. For instance, if you try to think of the purchase of particular delivery vehicles as directly causing the cost of delivery, you will get into trouble because delivery costs are also affected by wage scales paid to drivers, the distances the trucks are driven, and in fact, the number of deliveries that are made. Each of these things influences the cost of delivery, but none of them controls it. Similarly, in Figure 7, many of the results are influenced by both actions, and if we added the action "adjust frequency of telephone calls" it would influence many of these results also.

At this stage of the development of the conceptual model, we must be careful not to insert results that are too far-fetched. Moreover, we must recognize that some factors are influenced directly by a managerial action, and some are influenced only indirectly. For instance, almost all managerial actions have an influence on profit. Similarly, a number of managerial actions influence sales volume. To include these as results will be helpful for only a few managerial actions. We should concentrate on direct results. For instance, some managerial actions, such as "select products to sell," may directly affect the size of the market in which the firm competes. Others, such as "adjust advertising expenditures," may directly influence the share of the market that the firm enjoys. These actions have an indirect or derived effect on sales volume.

The results of managerial actions do not exist in a void. They influence one another also. For instance, to continue with the last example, sales volume is influenced directly by the size of the market and the penetration of the market. With diligence and care, we can determine which results are influenced by which other results, and, in total, we can develop a structure of results. Figure 8 represents a por-

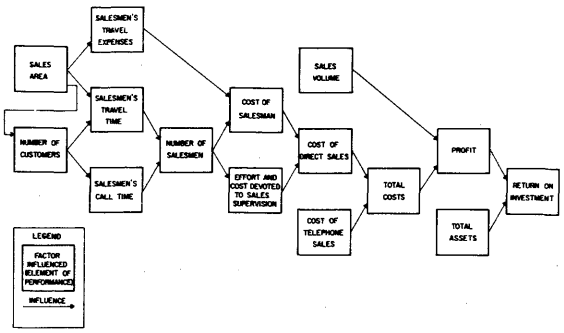


Figure 8. Structure of Results (Partial and Simplified).

tion of such a structure. It is drawn in flowchart form, and it has a natural progression from left to right. Observe that sales area is the most causal, basic element of performance, and return on investment is the most final, common element.

The preparation of this structure of results will assist in simplifying the statements of results of each managerial action. From the structure, we can infer that any one result has a chain of influences. Since we have the structure, it would be redundant to repeat the chain for each managerial action. It is enough to note the left-most element in the chain as a result of an action. For instance, Figure 7 does not show number of salesmen as a result.

Furthermore, the very exercise of trying to compile a complete structure of all results of managerial actions is likely to point out some results that have not been linked to any action. If the result stands at the beginning of a chain, we should try to find the managerial action that influences it.

Figure 9 shows the same structure of results as does Figure 8, but superimposed on it are the managerial actions that influence the results. This exercise can help us to understand how the results of one action can influence another action. It can also help us to see the managerial actions that are influenced by specific factors. For instance, the action "adjust working hours of salesmen" is affected by "salesmen's travel time" and "call time."

The flowchart is a very cumbersome device to display a complex structure of results. Figures 8 and 9 are simple only because they depict so few factors. We might try to simplify the job by using a precedence matrix such as Figure 10. A primary advantage of a matrix form of documentation is that it permits us to say something about the nature of the relationship between an action and its results, and between various results. Some of these relationships are clearly defined. After all, some of them are taken almost directly from accounting practice, and are, therefore, susceptible to the accounting definitions. We know that some other relationships are proportional, even though we may not know what the exact proportion is. The inter-

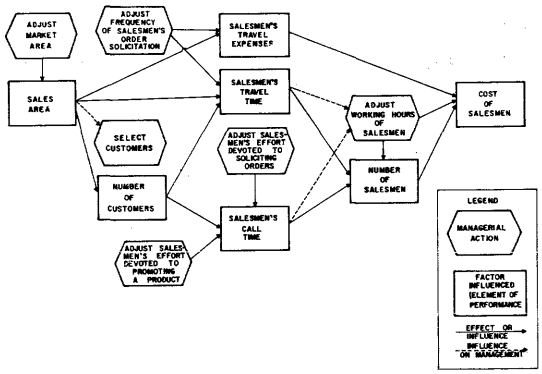


Figure 9. Managerial Actions Superimposed on the Structure of Results.

ACTION OR RESULT	Line No.	CORRESPONDING LINE NUMBER
ACTIONS		
Adjust Market Area	1	
Select Customers	2	
Adjust Salesmen's Effort - Promoting	3	
Adjust Salesmen's Effort - Soliciting Orders	4	
Adjust Frequency of Salesmen's Order Soliciting	5	
Adjust Working Hours of Salesmen	6	
RESULTS		
Sales Area	7	
Number of Customers	8	
Salesman's Travel Expenses	9	
Salesman's Travel Time	10	
Salesman's Call Time	11	
Number of Salesmen	12	
Cost of Salesmen	13	
Effort and Cost Devoted to Sales Supervision	14	
Cost of Direct Sales	15	
Cost of Telephone Sales	16	
Total Costs	17	
Sales Volume	18	
Profit	19	
Total Assets	20	
Return on Investment	21	

LEGEND: The "line" causes an effect on the "column" that is:
A - Arithmetic
F - A Complex Function
P - Proportional
U - Unknown

Figure 10. Matrix of Cause-Effect Relationship.

sections of the matrix can contain all that we know about the nature of the relationship. If we could determine the exact mathematical function that relates each of the actions to its results and the results one to another, we would have a fabulous mathematical model of the firm. Unfortunately, the nature of many of the relationships is simply unknown.

This exercise completes the conceptual model of the firm, its management actions, and the results of those actions. The model can be used as a general guide to understanding how the firm works. It might be used as the basis for a mathematical model of the firm.

DETERMINING INFORMATION CONTENT

The major purpose of creating the model in the first place was to assist in determining the information that is needed by the management of the company to manage the company well. This can be done by simply reviewing the action/result models. (See Figure 7.) We can consider each element of the model as a requirement for managerial information. We would like to measure the managerial actions themselves—how much action is taken, when was it taken, etc.—and we would like to measure each of the results of the action. A comprehensive information system will contain each of these measurements. In addition, it will contain many similar measurements of competitors' business practices.

In order to be more specific, we need to return to the diagrams of actions and results. One of the actions in Figure 7 is "adjust frequency of salesmen's order solicitation." This automatically suggests the question, "How often do salesmen solicit orders?" The simplest answer to that question is the total number of calls made by all salesmen in a month. Of course, we might want a finer breakdown—number of calls made by each salesman, and number of calls made on each customer. A tally of the number of calls is a frequency from the firm's point of view, but we might want to turn it around and look at it from the customers' point of view. How many solicitations does the average customer receive in a month?

Call frequency is not hard to measure. In fact, if we tried a little bit, we might even be able to learn how frequently our competitors solicit orders. If we asked our customers who else they buy from, and how frequently each competitor's salesman calls, we can expect some customers to refuse to answer, and some customers to give us wrong answers. But if we carefully compile the data that we do get, we can expect to be better informed than we would have been otherwise.

If we dwell on the subject longer, we might think of some other significant measures of the action itself, but we should also be concerned with the results of the action. "Salesmen's travel time" would be fairly easy to measure. All we need to do is ask the salesmen to keep track, for a month, of the time of day at which they leave one account and arrive at another account. We might even ask the salesmen to take an hour or so and prepare a "typical" itinerary with an estimate of the travel time between accounts. Either way, this is not an onerous chore, and it might even be worthwhile for the salesmen to go through the exercise just for what they would learn from it. Then we will need to compile it to learn the travel time.

"Salesmen's travel expenses" are regularly measured by most firms. We should observe in passing that there is a close connection between travel time and travel expenses. Furthermore, both factors, time and expenses, cannot be attributed to individual customers. Any attempt to determine the amount of travel time or travel expense that is incurred on behalf of any one customer is bound to be arbitrary. Neither of these factors lend themselves to interpretation from the customers' point of view.

If possible, we would like to go beyond measuring the action and its results. We would also like to consider the characteristics of the relationship between them. For these particular actions and results, we have a pretty good notion of the basic relationship—as the calling frequency is increased, the travel time and expense increase also. Any information beyond this intuitive feel will be difficult to acquire. We might ask a few salesmen to play a game with us and prepare hypothetical itineraries for the manner in which they would cover their territories if they were to cut their number of calls

to one-third of their present frequency. Then do it again for two-thirds, three-halves, and double. A compilation of these estimates should give us pretty good information about the relationship between the action, "adjust frequency of salesmen's order solicitation," and the results ,"salesmen's travel time" and "salesmen's travel expenses."

Another important characteristic of information is that it must be related in time, and in many cases, it must be understood "through" time. Each action and each result must be thought of as a time series. We want more information than just the present status. We also want to know how frequently we called on customers last year and the year before; and we want to know what frequency is planned or expected in the future. In addition, we want comparable information about travel time and expenses. If we can get nothing better, we might even use an historical comparison to tell us about the relationship between call frequency and travel time and expense.

The process of defining information requirements—the content of an information system—is to find a way to measure each managerial action, each result, and each connection between an action and a result. Then see if a comparable measurement can be found for competitors. We must be certain that the information can give an historical perspective and a glimpse of the future. In many cases, this method will lead us to unexpected information requirements.

How many sales managers do you know who could give you a satisfactory, quantitative answer to the question, "How often do salesmen solicit orders?" Most management information systems pass up this information completely, and yet, if we have any faith in our model, we can see that the action that is measured by the answer to that question has a far-reaching effect upon salesmen's time and expenses, and upon sales volume.

The job of translating these information requirements into reports and files is no small job, but it is a more familiar one. Systems and procedures people have been doing this sort of thing for years. Anyway, we have not yet found a way to have conceptual models help with this part of the job.

CONSTRUCTING CONCEPTUAL MODELS

The procedure for developing a complete conceptual model is easy to work with mentally. It progresses from one stage to the next, and at each stage we can focus our attention on only a few factors at a time. In the early stages, these factors are abstract. They are so abstract that they can apply with little modification to a number of different economic enterprises. But, as the early framework is expanded and completed with more details, the conceptual model begins to apply only to the economic enterprise for which it is designed.

I would not mean to imply by these words that conceptual models are easy to develop. It is one of the most rigorous mental exercises that I have run into. To complete a model requires creativity, imagination, insight, and judgment. I firmly believe that no one person can construct a good conceptual model. The best way is to develop one through individual effort which is followed up with a review by one or more persons. If this review is not available, the next best alternative is to attempt to complete several stages of development of a model. Then, put it away in a desk drawer and come back to it in six months. By this time, you may be a different enough person to review your own work adequately.

Don't get fooled by all the flow charts and geometric shapes. They are not the conceptual model. The model exists in the mind. The lines, words, and shapes are only a means of communicating and permanently recording what the mind has conceived.

Recall that we are dealing with a model, and a model is something that simplifies reality. The model does not faithfully reproduce every attribute and characteristic of the original; if it did, it would be a duplicate not a model. A wind tunnel model attempts to reproduce the exterior shape of an airplane or flying object so that engineers can observe the performance of the shape in moving air. A mathematical model for inventory control does not reproduce all the characteristics of the real world; it reproduces only those characteristics which are

felt to be of primary importance in controlling inventory. This same sort of attention must be applied to conceptual models for determining information requirements. The developer must continually weed out and separate trivial details from important generalities. For instance, back in Figure 7, we might have shown "number of salesmen's direct orders" as an element of performance, but we cannot find anything significant about that number.

At each stage of the development of a model, the analyst should ask himself: "Is each of the elements or factors which I have written on this page of approximately equal importance?" Since there is no absolute scale of importance, this question cannot be answered conclusively. That is why our model is a conceptual one. It deals with words, abstractions, and impressions. As such, it is subject to arbitrariness and judgment. Even so, it is worth developing.

REVIEWERS, PANELISTS, AND SESSION CHAIRMEN

AFIPS and the 1964 Spring Joint Computer Conference Committee would like to express their sincere appreciation to those listed below for their contribution toward the formulation and execution of the technical program.

REVIEWERS

M. ADELSON
J. P. ANDERSON
B. ARDEN
G. ARNOVICK
P. BAGLEY
J. BULGER
R. G. CANNING
T. E. CHEATHAM
E. G. CLARK
M. CONNELLY
J. E. CREMEANS
E. P. DAMON
W. DORFMAN
A. C. DOWNING
K. EISEMANN
R. D. ELBOURN
P. L. GARVIN
J. T. GODFREY
M. H. HALSTEAD
J. HAMBLEN
H. HELLERMAN
W. R. HOOVER
W. A. HOSIER
R. M. HOWE
E. T. IRONS
L. KANENTSKY
H. H. LOWELL
A. P. MACFARLAND
J. H. MACLEOD
M. S. MASON
E. M. MCCORMICK
T. MCFEE
M. MELKANOFF
D. M. MICHAEL
J. MINKER
G. L. MITCHELL
H. OSER
H. PETERSON
J. A. POSTLEY
A. RALSTON
L. C. RAY
A. ROBINSON
J. E. ROWE
B. SAMES
C. SHAW
H. K. SKRAMSTAD
A. E. SPECKHARD
T. B. STEEL
W. P. TIMLAKE
C. W. TURNER
H. VAN ZOEREN

PANELISTS

E. ADAMS
M. ADELSON
G. N. ARNOVICK
J. E. CREMEANS
A. E. DANIELS
W. DORFMAN
A. C. DOWNING, JR.
N. P. EDWARDS
P. L. GARVIN
H. HELLERMAN
W. R. HOOVER
W. A. HOSIER
R. M. HOWE
J. L. JONES
H. H. LOWELL
E. J. MAHONEY
R. P. MAYER
E. MCCLOY
J. MCLEOD
D. N. MICHAEL
A. E. MILLER
J. L. MITCHELL
J. A. POSTLEY
A. RALSTON
J. C. SHAW
H. K. SKRAMSTAD
A. E. SPECKHARD
N. STATLAND
T. B. STEEL, JR.
F. B. THOMPSON
W. P. TIMLAKE
R. VAN HORN
A. N. WILSON

SESSION CHAIRMEN

W. F. BAUER
R. DAVIS
B. A. GALLER
H. L. GELERNTER
C. C. GOTLIEB
R. W. HAMMING
R. M. HAYES
J. MOSHMAN
W. PAPIAN
J. D. PORTER
L. C. RAY
J. E. SHERMAN
J. SINGLETON
T. D. TRUITT

AMERICAN FEDERATION OF INFORMATION PROCESSING SOCIETIES (AFIPS)

211 E. 43rd Street, New York 17, New York

Chairman

MR. J. D. MADDEN
IBM Corporation
P. O. Box 390
Poughkeepsie, New York

Chairman-elect

DR. EDWIN L. HARDER
Westinghouse Electric Corporation
700 Braddock Avenue
East Pittsburgh, Pennsylvania

Secretary

MR. SAUL I. GASS
IBM Corporation
1800 Yosemite Road
Berkeley 7, California

Treasurer

MR. FRANK E. HEART
Lincoln Laboratory
P. O. Box 73
Lexington 73, Massachusetts

Executive Committee

DR. ARNOLD A. COHEN, IEEE
MR. WALTER M. CARLSON, ACM
MR. FRANK E. HEART, IEEE
MR. CLAUDE A. R. KAGAN, IEEE

Executive Secretary

MR. REDMOND S. GARDNER
AFIPS Headquarters
211 E. 43rd Street
New York 17, New York

ACM Directors

MR. H. S. BRIGHT
Philco Corporation
Computer Division
3900 Welsh Road
Willow Grove, Pennsylvania

MR. EUGENE H. JACOBS
System Development Corporation
2500 Colorado Avenue
Santa Monica, California

MR. WALTER M. CARLSON
Director Technical Information
Office DDR&E
Office Secretary of Defense
Washinfton 25, D. C.

DR. ALAN J. PERLIS
Department of Mathematics
Carnegie Institute of Technology
Pittsburgh, Pennsylvania

IEEE Directors

MR. WALTER L. ANDERSON
General Kinetics, Inc.
2611 Shirlington Road
Arlington 6, Virginia

IEEE Directors

DR. ARNOLD A. COHEN
Univac Division
Sperry Rand Corporation
St. Paul 16, Minnesota

MR. G. L. HOLLANDER
Hollander Associates
P. O. Box 2276
Fullerton, California

MR. CLAUDE A. R. KAGAN
Western Electric Company
P. O. Box 900
Princeton, New Jersey

SCI Member of Governing Board

MR. JOHN E. SHERMAN
Lockheed Missiles and Space Company

Standing Committee Chairmen

Admissions
MR. BRUCE GILCHRIST
Service Bureau Corporation
425 Park Avenue
New York 22, New York

Awards
MR. SAMUEL LEVINE
Teleregister Corporation
445 Fairfield Avenue
Stamford, Connecticut

Conference
MR. KEITH UNCAPHER
The RAND Corporation
1700 Main Street
Santa Monica, California

Constitution & By Laws
MR. EUGENE JACOBS
System Development Corporation
2500 Colorado Avenue
Santa Monica, California

Finance
DR. R. R. JOHNSON
General Electric Company
Computer Division
13430 North Black Canyon Highway
Phoenix, Arizona

International Relations
MR. I. L. AUERBACH
Auerbach Corporation
1634 Arch Street
Philadelphia 3, Pennsylvania

Planning
MR. G. L. HOLLANDER
Hollander Associates
P. O. Box 2276
Fullerton, California

Social Implications of Information Processing Technology
DR. MORRIS RUBINOFF
Moore School of Electrical Engineering
University of Pennsylvania
Philadelphia 4, Pennsylvania

Public Relations
MR. ISAAC SELIGSOHN
IBM Corporation
590 Madison Avenue
Department 1-812
New York 22, N. Y.

Education
DR. RICHARD M. BROWN
University of Illinois
Coordinated Science Laboratory
Urbana, Illinois

1964 SPRING JOINT COMPUTER CONFERENCE COMMITTEES

Chairman
HERBERT R. KOLLER, U.S. Patent Office

Vice-Chairman
ALEXANDER C. ROSENBERG, Goddard Space Flight Center

Secretary
JOSEPH O. HARRISON, JR., Research Analysis Corp.
RICHARD G. WILLIAMS, Research Analysis Corp., Alternate

Technical Program
JACK ROSEMAN, C-E-I-R, Inc., Chairman
DOMINIC A. LEITI, C-E-I-R, Inc., Vice Chairman
BERNARD COHEN, C-E-I-R, Inc.
HOWARD E. TOMPKINS, University of Maryland
G. H. SWIFT, I.B.M.
AUTHUR I. RUBIN, Martin Aircraft Corp.
JACK MINKER, Auerbach Corp.
ELSIE M. MAMO, Computer Dynamics, Inc.

Exhibits
SOLOMON ROSENTHAN, U.S.A.F., Chairman
GEORGE HOPPING, General Services Administration, Co-Chairman

Printing and Mailing
MIKE HEALY, System Development Corp., Chairman
LOUIS ELIAS, System Development Corp., Co-Chairman

Registration
JOSEPH H. EASLEY, UNIVAC, Chairman
NORMAN G. YOUNG, UNIVAC
JAMES LUNGWITZ, UNIVAC

Public Relations
J. HUGH NICHOLS, Dunlap and Associates, Inc., Chairman
JOHN E. KUMPF, UNIVAC, Co-Chairman
JOHN L. REYNOLDS, I.T. & T., Co-Chairman

Hotel Arrangements
CLARK J. RISLER, Litton Industries, Chairman
RICHARD H. SMITH, Control Data Corp.
PAT DOYLE, National Bureau of Standards
MARY L. DOUGLAS, Applied Physics Laboratory

Finance
RICHARD C. LEMONS, General Electric Co., Chairman
NICHOLAS J. SUSZYNSKI, JR., General Electric Co., Co-Chairman

Field Trips
JOHN J. GLYNN, Defense Documentation Center, Chairman
EDWARD J. CUNNINGHAM, Air Force Systems Command, Co-Chairman

Proceedings
GORDON D. GOLDSTEIN, Office of Naval Research, Chairman
MARGO A. SASS, Office of Naval Research, Co-Chairman

Ladies Activities
RENEE JASPER, Auerbach Corp., Chairman
IBY HELLER
JUDY ROSEMAN
IDA GOLDSTEIN
RITA MILLER
SALLY PEAVY

Consultants
MARTIN S. BECKER, Legal Counsel
JOHN HOSKINS, Graphic Design
COMPTON JONES ASSOCIATES, Public Relations
JOHN C. WHITLOCK ASSOCIATES, Exhibits

EXHIBITORS
1964 SPRING JOINT COMPUTER CONFERENCE

AB ATVIDABERGS INDUSTRIER
ACADEMIC PRESS, INC.
ADAGE, INC
ADDRESSOGRAPH-MULTIGRAPH CORPORATION
ADVANCED SCIENTIFIC INSTRUMENTS—DIVISION OF EMR
AERONUTRONIC DIVISION—PHILCO CORPORATION
AMERICAN DATA PROCESSING INC.
BURROUGHS CORP., ELECTRONIC COMPONENTS DIV.
CALIFORNIA COMPUTER PRODUCTS, INC.
CAMBRIDGE COMMUNICATIONS CORP.
C-E-I-R
CHRONO-LOG CORPORATION
COLLINS RADIO COMPANY
COMMERCE CLEARING HOUSE, INC.
COMCOR, INC.
COMPUTERS AND DATA PROCESSING MAGAGZINE
DATAMATION MAGAZINE
DATA SYSTEMS DESIGN MAGAZINE
DATA PRODUCTS CORPORATION
DI/AN CONTROLS, INC.
DIGI DATA CORPORATION
DIGITAL EQUIPMENT CORPORATION
DIGITRONICS CORPORATION
DYMEC
DIVISION OF HEWLETT-PACKARD CO.
GENERAL KINETICS, INC.
GPS INSTRUMENT COMPANY, INC.
HONEYWELL EDP DIVISION
IBM CORPORATION
INDIANA GENERAL CORPORATION
INTERNATIONAL COMPUTERS AND TABULATORS, LTD.
INVAC CORP.
ITT DATA PROCESSING CENTER
MILGO ELECTRONIC CORP.
THE NATIONAL CASH REGISTER COMPANY
NAVIGATION COMPUTER CORP.
OHR-TRONICS, INC.
OMNITRONICS, INC.
PACKARD BELL COMPUTER
PHILCO CORPORATION — SUBSIDIARY OF FORD MOTOR COMPANY (INFORMATION SYSTEMS)
PHOTOCIRCUIT'S CORPORATION
RMS ASSOCIATES, INC.
ROYAL McBEE CORPORATION
SCIENTIFIC DATA SYSTEM, INC.
THE SERVICE BUREAU CORPORATION
SOROBAN ENGINEERING, INC.
STATISTICAL TABULATING CORP.
SYSTEM DEVELOPMENT CORP.
TALLY CORP.
TELETYPE CORPORATION
AMERICAN TELEPHONE AND TELEGRAPH COMPANY
AMPEX CORPORATION
ANELEX CORPORATION
APPLIED DYNAMICS, INC.
BECKMAN INSTRUMENTS, INC.
BENSON-LEHNER CORP.
BRYANT COMPUTER PRODUCTS
COMPUTER DESIGN
COMPUTER SYSTEMS, INC.
COMPUTERS AND AUTOMATION
COMPUTER CONTROL COMPANY, INC.
COMPUTER SCIENCES CORPORATION
CONTROL DATA CORPORATION
COOK ELECTRIC COMPANY—DATA-STOR DIVISION
CORNING GLASS WORKS
CYBETRONICS, INC.
DATAMEC CORPORATION
ELECO CORPORATION
ELECTRONIC ASSOCIATES, INC.
ENGINEERED ELECTRONICS COMPANY
ELECTRONIC MEMORIES, INC.
FABRI-TEK, INC.
FERROXCUBE CORPORATION OF AMERICA
GENERAL DYNAMICS/ELECTRONICS—GENERAL ELECTRIC COMPUTER DEPARTMENT
LIBRASCOPE GROUP, GENERAL PRECISION, INC.
LOCKHEED ELECTRONICS COMPANY—AVIONICS AND INDUSTRIAL PRODUCTS DIVISION
MAC PANEL COMPANY
MEMOREX CORPORATION
MIDWESTERN INSTRUMENTS, INC.
McGRAW-HILL BOOK CO., INC.
MONROE CALCULATING MACHINE COMPANY, COMPANY, INC.—DIVISION OF LITTON INDUSTRIES
POTTER INSTRUMENT COMPANY, INC.
PRENTICE-HALL, INC.
RADIATION, INC.
RAYTHEON COMPANY
RCA — EDP DIVISION
RCA ELECTRONIC COMPONENTS AND DEVICES
RECORDAK CORPORATION
RFS ENGINEERING COMPANY
SPARTAN BOOKS, INC.
TEXAS INSTRUMENTS, INC. — INDUSTRIAL PRODUCTS GROUP
TRANSISTOR ELECTRONICS CORPORATION
UGC INSTRUMENTS — DIVISION OF UNITED GAS CORPORATION
UNIVAC DIDVISON OF SPERRY RAND CORP.
UPTIME CORP.
WESTINGHOUSE ELECTRIC CORP.
JOHN WILEY & SONS, INC.
WYLE LABORATORIES

AUTHOR INDEX