Reprinted from BEHAVIORAL SCIENCE, Vol. 7, No. 1, January 1962

The Use of Simplified Programming Systems in IBM 650 Data Processing

Linton C. Freeman, Syracuse University Computing Center.

Computers are no longer the esoteric playthings of mathematicians and engineers; they have become the day-in-day-out workhorses of data analysis. As they get wider application, there is increasing demand for the production of programs for the solution of new problems. But the number of programmers who are able to prepare all these new programs is limited.

This problem is compounded by the fact that efficient machine use requires that the program writer be thoroughly familiar both with programming procedures and with the details of the problem at hand. Since most programmers are trained in mathematics or engineering, this condition raises no special problems for the mathematical or engineering use of the computer. For the behavioral scientist, however, it often creates some additional difficulties.

Very few individuals are trained both in computer programming and in a behavioral science discipline. So behavioral scientists often have trouble in communicating the details of their data processing problems to programmers, and programmers are unable to find behavioral scientists who are willng and able to learn the intricacies of programming.

What is needed are individuals trained in both areas; in a behavioral science and in computer programming. In the long run, the best policy might be to encourage—even require—behavioral science graduate students to develop at least the most basic skills in programming. It is fairly clear that such skills will be important to future generations of behavioral scientists. But in the meantime, some quicker means of meeting our immediate needs must be found.

For the most part, people who are actually conducting research in the various behavioral sciences have neither the time nor the background to become specialists in programming. Altogether too much time is required, both to learn the basic skills and to apply them to problems as they arise. Some sort of short-cut must be employed; and fortunately such a short-cut is available in the form of various simplified programming systems which are easy to learn and quick to apply.

The number of simplified programming systems which have been developed is legion. For the most widely used computer, the IBM 650, there are twenty or thirty interpretive systems and compilers in general use. Most statistical programs are prepared according to one or another of these simplified schemes, but the choice among them is often a matter of personal taste on the part of the writer. The various programming systems, however, have different strengths and weaknesses; a choice among them should be made in light of the nature of the problem rather than on the basis of some arbitrary whim of the programmer.

Some writers have explored the details of a few of these programming systems (Curtz, Riordan, & Spohn, 1960; Arden & Graham, 1959; Kanner, 1959), but no general rules for choosing among them have been developed. The present paper represents an attempt to explore some of the problems of choosing among a number of programming systems for the IBM 650. Several systems will be compared in terms of their efficiency in handling various phases of a typical problem of statistical data analysis which might arise in a behavioral science setting.

Three factors might be considered in choosing a programming system for a particular problem: (1) the anticipated number of runs for which the program is to be used, (2) the relative amount of input and output as compared with the amount of internal machine computation involved in the problem, and (3) the availability of machine time for solution of the problem. By taking these three factors into account we are provided with a rational basis for choosing a programming system in a particular case.

The problem selected for study is typical of those confronted in statistical data analysis. It involves the computation of chi-square as a test of significance in a 2 X 2 con-

1 The preparation of this report was supported by research grant NSF-G14S94 of the National Science Foundation.

tingency table. Such a problem involves a relatively large amount of input and output and employs a straightforward series of arithmetic operations. The logic is simple, and there is no need for any extensive iterative processes. From analysis of a problem of this sort we can infer some of the consequences of using various programming systems for the solution of many similar statistical analyses.

All in all, seven versions of the chi-square program were prepared for the IBM 650 computer. Two of these programs were written in the basic 650 machine language and five were prepared according to simplified systems. Machine language is relatively more difficult to learn and extremely tedious to write. The other systems are all easier to learn, and all of them speed up the writing process.

The two machine language programs were prepared in order to provide a basis for comparison with the simplified systems. One was a hand-optimized version written for the regular drum storage unit of the computer, and the other was written for the immediate access storage unit. The immediate access unit has a small capacity and cannot be used for longer programs, but it represents the fastest possible program for the 650. It is the ideal which is approximated by the other programs.

Simplified programming systems may be roughly categorized according to their principle of simplification. Basically, they may be classified into two types: interpretive systems and compilers. An interpretive system is a superprogram, written in machine language and stored in the machine, which causes the computer to read simplified instructions and translate them into its own language. Each instruction is read, interpreted, and executed; thus, in effect, the computer is programmed to simulate another, simpler, machine.

A compiler is a program which causes a computer to read a set of simplified instructions, translate them into machine language, and prepare a new set of instructions in machine language which may then be used to solve problems. In an interpretive system each instruction is translated each time the program is run; in a compiler the translation occurs only once, before the program is run.

Although there are many interpretive systems and compilers available, only a few of them are in general use. In this analysis an attempt was made to select five of those most commonly encountered and to compare them. Two interpretive systems and three compilers were chosen. The interpretive systems were the Bell Interpretive System (Wolontis, 1956) and the Revised Bell Lab Interpretive System (Hall, undated). The compilers were GAT (Arden, undated), IBM 650 Fortran (650 Fortran, 1960), and the Symbolic Optimum Assembly Program, SOAP (SOAP II, 1957).

Each of these systems represents a simplification of machine language. It is difficult to say which is easiest to prepare, but Fortran must be considered a strong contender for that position. Writing Fortran programs is almost exactly like writing down algebraic expressions. GAT is similar, but it places more restrictions on the use of symbols. The two Bell System programs probably rank next; they are machine languages, but simple ones. And SOAP is closest to actual 650 machine language, but still a great deal less tedious to produce. It is likely that very few programmers would argue with the rankings of these systems in terms of difficulty of program preparation listed in Table 1.

TABLE 1
PROBABLE DIFFICULTY OF PROGRAM PREPARATION OF SEVERAL PROGRAMMING SYSTEMS

Programming System	Level of Difficulty
Machine Language	5
SOAP	4
Bell Systems	3
GAT	2
Fortran	1

When a compiler is used in preparing a program, the program must be run through the machine in order to translate it into machine language. Some compilers require one initial pass, but others entail two or even three. Even when programs are prepared by means of interpretive systems or in machine language an initial pass is customary. This is because typically such programs are written one instruction per card. If they are left in that form they are slow to load, so one-per-card programs are usually run into the computer and punched out at six or seven instructions per card. Thus their loading speed is increased for subsequent data processing runs. In any case, an initial assembly run is customary in order to provide an efficient final program. The assembly times for the chi-square program are shown in Table 2.

TABLE 2
ASSEMBLY TIMES FOR THE VARIOUS PROGRAMMING SYSTEMS FOR THE CHI-SQUARE PROBLEMS

Programming System	Time (in seconds)
Fortran	427
GAT	165
SOAP	146
Bell System	78
Revised Bell System	78
Machine Language (Drum)	4S
Machine Language (Core)	4S

Table 2 shows that the order of the assembly times for these programs is roughly the inverse of their programming difficulty. The only exceptions are the Bell Systems. They are easier to write than SOAP programs, and yet they assemble more quickly. This is because SOAP is a compiler—its assembly phase includes translation into machine language, while Bell System programs are translated later during computation. We might therefore expect SOAP to produce a more efficient program.

Once a program has been assembled it is ready to run. It may be used in the solution of problems, but first it must be read into the machine. The various program read times are listed in Table 3.

Here the SOAP and machine language programs are outstanding. Fortran and GAT are slower since they both require that standard subroutines be read in along with the program. And the Bell Systems require that an interpretive program be included which will allow the computer to translate each Bell instruction into machine language.

TABLE 3
PROGRAM READ TIMES FOR THE CHI-SQUARE PROBLEM

Programming System	Time (in seconds)
Revised Bell System	72
Bell System	65
GAT	52
Fortran	33
SOAP	7
Machine Language (Drum)	5
Machine Language (Core)	5

Once the program is loaded data can be entered and problems solved. Table 4 shows the number of chi-square problems which can be solved per minute using each of these programming systems.

TABLE 4
NUMBER OF CHI-SQUARE SOLUTIONS PER MINUTE FOR VARIOUS PROGRAMMING SYSTEMS

Programming System	Number of Runs (per minute)
Bell System	39
GAT	49
Fortran	58
Revised Bell System	61
SOAP	100*
Machine Language (Drum)	100*
Machine Language (Core)	100*

* These represent limits imposed by the output unit of the computer. A modification of the program output form could speed up these programs.

The SOAP and machine language programs turn out to be faster than the others. But since their output was restricted by a machine limitation, a test was made by running the programs without their read and punch instructions.

TABLE 5
NUMBER OF CHI-SQUARE SOLUTIONS PER MINUTE WITHOUT READ OR PUNCH INSTRUCTIONS

Programming System	Number of Runs (per minute)
Bell System	55
Revised Bell System	84
GAT	220
Fortran	326
SOAP	341
Machine Language (Drum)	494
Machine Language (Core)	930

The results listed in Table 5 indicate the actual computation speed of these systems. Given this information, there is no doubt that if rate of computation were the only consideration, core machine language would be the only choice. The core, however, has an extremely small capacity, and can be used only when the problem is not too large. Furthermore, machine language is a great deal more difficult to write than other systems, so its use might not always be economical. But the choice among the simplified systems is not always obvious. The tables above reveal strengths and weaknesses in each of the simplified programming systems. In any given situation a choice among them can be made in terms of these strengths and weaknesses.

Since they require less computer time both for assembly and program running, the machine language systems seem to be called for whenever machine time is at a high premium. Then, too, if a program is to be used quite regularly and over an extended period of time, the greater speed of machine language programs makes them desirable. In other cases, the simplified systems seem satisfactory.

If we compare the machine language systems with SOAP, which is considerably easier to use, it becomes apparent that unless the program is going to be used almost all day every day, SOAP will be satisfactory. In a situation where machine time is relatively unavailable, or when a program is to be run quite often, SOAP seems to be the choice among the simplified systems. This is particularly true when there is a great deal of input and output. In such a case SOAP, with its relatively greater read and punch speed, would provide more output.

Fortran would be a good choice only when a great number of runs could be anticipated. Its excessive assembly time precludes its use for one-shot programs. Furthermore, its slow rate of input and output suggest that it might best be employed on problems which have little input and output but a relatively large amount of internal computation.

In contrast, GAT assembles and accepts input and produces output relatively quickly, but its rate of computation is slower. This suggests that the appropriate application of GAT is for problems involving a small amount of internal computation, when only one or a few runs are anticipated.

Finally, the original Bell System seems wholly useless, and the Revised Bell System might best be employed for problems involving vast amounts of input and output, relatively little internal computation, and few runs. It is able to handle input and output faster than GAT or Fortran, but its rate of computation is considerably slower.

REFERENCES

Arden, B. W. *Generalized algebraic translator* (GAT), IBM 650 Program Library, 2.1.007, Undated.

Arden, B., & Graham, R. On GAT and the construction of translators. *Communications Assoc. Computing Machinery,* 1959, 2, 24-26.

Curtz, T. B., Riordan, J. F., and Spohn, M. A comparison of 650 programming methods. *Communications Assoc. Computing Machinery,* 1960, 3, 663-671.

Hall, D. J., *Revised Bell Laboratory interpretive system.* IBM 650 Program Library, 2.0.015, Undated.

Kanner, H. An algebraic translator. *Communications Assoc. Computing Machinery,* 1959, 2, 19-22.

650 Fortran—automatic coding system for the IBM 650 data processing system. *IBM Reference Manual,* 1960.

SOAP II for the IBM 650 data processing system. *IBM Reference Manual,* 1957.

Wolontis, V. M. A complete floating decimal interpretive system for the IBM 650 magnetic drum calculator. *IBM Technical Newsletter,* 1956, 11.

(Manuscript received April 28, 1961)