A SIMSCRIPT-FORTRAN Case Study

ARLA E. WEINERT*
Research Analysis Corporation, McLean, Virginia

Two programs for a vehicle dispatching model, one written in 7040 SIMSCRIPT and the other in 7040 FORTRAN IV are compared. The comparison is made in terms of basic program design decisions, storage requirements, computer time used, and the ease of making changes.

In the SIMSCRIPT program, the primary design considerations center around the choice of model variables, model changing events, and model testing. In the FORTRAN program, basic design problems relate to the representation of the passage of time, the allocation of storage, and the organization of input data.

The comparison of these differently designed programs shows that the SIMSCRIPT program uses more computer storage and more computer time, but requires fewer program changes to introduce model revisions.

Comparisons have been made of the features provided in different systems designed to handle digital simulation problems [1–4]. There is also a comparison of two matched codes done in different simulation systems [5]. These comparisons offer insight into the characteristics of simulation problems that have received special attention. For a practical problem, however, the choice of a computer language may depend on what is available on the computer to be used and who knows how to use it.

The person with a simulation problem may find that the computer most readily available has no simulation language. Or he may find that the people working on the problem are familiar only with the general algebraic language used on the available computer. To provide an indication of the factors that should be considered in making a decision in such a situation this paper examines both a SIMSCRIPT and a FORTRAN program that implement the same vehicle-dispatching model on the IBM 7040 computer. After an initial examination of the difference in approach to the problem afforded by the two languages, three quantitative comparisons of the programs are made. These comparisons are of computer storage, computer time, and number of code changes needed to introduce model changes. Finally, these comparisons are discussed in terms of their implications for digital simulation problems in general.

* Present address: Naval Command Systems Support Activity, Washington Navy Yard, Washington, D.C. 20390.

The Model Simulated

The vehicle-dispatching model represented by the SIMSCRIPT and FORTRAN programs that are compared provides for the arrival of requests for missions to be performed by vehicles at arbitrary times during a simulated day. All vehicles in the fleet are of the same type. If a vehicle is available, it is assigned to each request when it arrives. Once a vehicle is assigned to a request, the mission is assumed to be successfully completed. On its return from a mission a vehicle may receive unscheduled and/or scheduled maintenance. At the completion of the mission and any attendant maintenance, a vehicle is available for assignment to a new request.

If a request arrives and no vehicle is available to meet it, the request waits until a vehicle becomes available. Requests that are delayed have vehicles assigned to them in the order the requests arrived. At the end of each simulated day all unmet requests are cancelled.

The user is given two mutually exclusive options for specifying request arrivals. He may treat them as exogenous events that happen outside the model simulated or he may let the model generate them as endogenous events. Treating the arrival of a mission request as an exogenous event means the user specifies the arrival time of each request and the type of mission requested. If desired, this makes it possible to put into the model an actual history of individual requests taken from field data. When the model is used to generate the mission requests, it assumes a Poisson distribution of arrival time for which the user supplies the parameters. The arrival times for individual requests are drawn at random from this distribution. The type of mission requested is then drawn at random from a user supplied probability distribution of expected mission types.

Two optional printouts are provided. In one the user can ask for a printout of the initial data used to start the operation of the model. The other can be obtained at the end of each simulated day. It shows the state of each individual vehicle and gives summary data on mission requests received, delayed, and completed. In addition

to these two optional printouts, summary information on vehicle-operating time; maintenance time; delays encountered by requests; and the number of requests received, delayed, and met, are automatically printed out at the end of the number of simulated days specified as part of the input data. The purpose of this output data is to provide a basis for the comparison of different types of vehicles when meeting different mixes of mission requests.

The vehicle-dispatching model can be thought of as a typical pilot study. The model is not trivial. On the other hand, it is not so large that it will obviously tax the memory capacity of current large scale computers. If its initial use provides useful insights into the functioning of an actual operating system, the model will be enlarged and revised to better portray some actual operating situation. This tendency of the model to grow is also typical of simulation problems.

Basic Program Decisions

On the IBM 7040, SIMSCRIPT is implemented as a FORTRAN pretranslator. It provides all the features of FORTRAN plus features particularly useful for digital simulation problems. One feature provided is a timing routine that keeps track of simulated time to 8 significant digits. This timing routine is used in conjunction with a scheme for scheduling events. These events are the means by which changes in the condition of the model simulated are effectuated. The state of the simulated model is described by the values of variables called "attributes." These variables describe the permanent or temporary "entities" that make up the simulated model. The permanent "entities" are parts of the model that exist as long as the simulated model exists. Temporary "entities" come and go as simulated time progresses and events produce changes in the simulated model. The events by which changes in the model are effectuated are a special kind of temporary entity that can be scheduled from within or outside the simulated model. Once the event takes place it no longer is a part of the simulated model.

SIMSCRIPT provides for identifying as members of a "set" entities that have specified common attributes. Entities may be filed in or removed from sets by specified rules. Sets may also be searched for entities meeting specified conditions. In addition to set manipulation, SIMSCRIPT also has statements that extend the logical capability of FORTRAN so that the user may apply arithmetic statements, logical conditions and search requests to all entities of a given kind. In addition, simple data gathering and statistic computing statements are provided.

Storage allocation is removed from initial programming considerations. It is deferred until the simulation is to be run. At that time the actual number of each kind of permanent entity to be used is specified. Provision is also made for storing only those temporary entities that are currently active in the simulated system. In case of a shortage of storage space, provision is made for packing up to four integer attributes in a computer word.

SIMSCRIPT also supplies a "Report Generator" that allows the user to layout output in the form he expects to see it. This is done by placing titles and headings on a form with 132-character lines as they are to appear on the report. This avoids the need for first doing the layout and then counting characters to translate the layout into suitable computer instructions.

There are other facilities available in SIMSCRIPT such as statements for the generation of random variables, the conversion of simulated time from one unit to another, and individual input or output statements some of which convert time variables automatically. All of these are desirable and convenient. The basic power of the language comes from the more general concepts discussed above. It is the implementation of these concepts in SIMSCRIPT that makes it possible to approach simulations in terms of problem content and, when necessary, to introduce changes relatively easily.

With these SIMSCRIPT facilities the basic questions considered in program development are of three kinds. First, what are the significant parts (i.e., "entities") of the simulated model and what variables (i.e., "attributes") describe them? Second, under what conditions do significant changes take place in the values of the variables that describe the simulated model? Third, what tests should be made to see that the computer program developed, accurately represents the model postulated?

The first two questions seem obviously related to specific SIMSCRIPT features. The first in effect concentrates on judicious selection of permanent and temporary entities and attributes to describe them. The second set of questions concentrates on the selection of simulated events and describing what happens in each event. The relation of SIMSCRIPT to the third set of initial considerations that deals essentially with program acceptance tests is not so obvious. It arises from the extensive SIMSCRIPT use of subprograms. Once an error-free SIMSCRIPT translation has been obtained, all these subprograms have a high probability of receiving legal input values. This means there is a high probability of successful program execution. Reliance on program hang-up to reveal mistakes becomes risky under these conditions. It is much better to build into the program, features that will make it easy to show how the programmed model actually works.

In the SIMSCRIPT program for the vehicle-dispatching model the first two kinds of basic program choices are simple. SIMSCRIPT features are well matched to the requirements of the vehicle-dispatching model. The entities are obviously vehicles (permanent) mission requests (temporary) and delayed request sets. A choice exists as to whether summary information to be accumulated should be kept separately by type of request and/or type of vehicle. In the vehicle-dispatching model it was decided to keep this information separate by type. This facilitates comparisons among vehicle types and types of mission

requests. The choice of organization for the delayed mission requests was between a first-in-first-out queue for each vehicle type and a single queue that can be searched on vehicle type, mission type, and arrival time. In the vehicle-dispatching model the latter was chosen. This choice was made on the basis of coding convenience even though it would mean longer execution times for model runs.

The events that are significant in the vehicle-dispatching model are also readily identified. There are the two kinds of mission request arrival events that have already been described. There is also an event that occurs when a mission is completed and a vehicle becomes available for assignment to a new request. The cancellation of unmet requests at the end of each simulated day is another model event. Two artificial events, one to start the model run and one to end it are also needed to control the operation of the model.

The choice of desirable program testing aids to build into the program is less obvious. The SIMSCRIPT program was supplied with what might be called a selective, time-dependent, optional model trace. This trace was dependent upon building into each event program an identifying code for the event and for each unique result the event could produce. Three printouts were also provided. One of these printouts was of the current random number as an octal number, a decimal integer, and as a SIMSCRIPT random floating point number. For all events a printout showing simulated time, the event code, and the event result code was supplied. For events that reflect changes in the condition of vehicles or mission requests, a printout is supplied that shows all attribute values at the end of the event for the affected vehicle, mission request, vehicle type and mission type as well as the delayed mission queue. All of these printouts are made optional on the basis of simulated time. The printouts needed for this can be produced easily with the SIMSCRIPT Report Generator. This scheme produces decimal printouts that anyone familiar with the model can examine to determine if the program behaves according to model specifications. The effectiveness of the scheme depends on the judgment used in placing the program switches that control the printouts. These switches allow the user to get the information he wants, and still not be inundated with printouts.

On the 7040, FORTRAN contains the familiar DO statements for controlling loops, logical IF statements with relational operators such as EQ for equal, LE for less than or equal, etc., input and output statements described by the usual FORMAT statements, and DIMENSION statements for reserving computer storage for one-, two-, and three-dimensional arrays. 7040 FORTRAN is similar to other versions of FORTRAN available on other computers. Its use on the vehicle-dispatching model can be considered representative of the use of an algebraic computer language on a digital simulation problem.

When using a general algebraic computer language without features specifically designed for simulation problems, the basic programming questions dealt with are different than those met when using SIMSCRIPT. One question that always arises is "How shall the passage of simulated time be represented?" Another question is "How shall computer storage be allocated to the various kinds of data used by the model?" A question that is very likely to arise is "How shall the input data be organized?"

If it is feasible to represent simulated time as a succession of time periods of equal length, a simple integer counter and the loop statements of an algebraic language can be adequate time keeping tools. In this kind of time keeping scheme, the computer program checks for all possible kinds of model activity and cycles through currently active phases of the model. The time period used should be long enough to prevent the program from cycling through many time periods with little or no model activity. It should also be short enough so that no significant model activity is lost. For the FORTRAN program for the vehicle-dispatching model, it was decided that the use of a 10-minute time period would adequately represent model activity.

Once a particular problem is stated, the organization and allocation of storage can often be tailored to fit the problem. One effect of this tailoring can be a sharp reduction in the amount of storage required. The organization of storage can also be used to help keep track of simulated time. This is done in the FORTRAN program for the vehicle-dispatching model. First, it was determined that anticipated use of the model would be confined to case with no more than 1000 vehicles, and 20 mission types. Comparisons would be made of up to 3 types of vehicles, but only one vehicle type would be active in the model at a time. It was also determined that no more than 36 mission requests would arrive in a 10-minute time period. With this information, a compact two-dimensional table of 144 rows and 36 columns was designed to represent the arrival of mission requests during a simulated day. In this table the 144 rows represent the 10-minute time periods. The 36 columns represent the order in which requests arrive during the 10-minute time period, and the value of the table entries represent the type of mission requested. When this table is supplemented by a row pointer and a column pointer that locate the last mission request to which a vehicle was assigned, this becomes a clever way to organize storage. This table design saves storage space. It also helps to keep track of simulated time, schedules the arrival of mission requests, and when the row pointer that helps to locate the last mission request serviced lags behind the current time period it serves as the basis for recognizing delayed mission requests. The remainder of the data for the model is stored in simple arrays or straight-forward two-dimensional tables. This kind of storage organization assigns all data to fixed computer storage locations. It is feasible for a competent programmer to use computer storage dumps to determine what has gone wrong in case the program fails to run. Otherwise the printouts required by the model of the input data, vehicle

TABLE I. 7040 Storage Utilization in the Vehicle-Dispatching Programs

Storage use	Core storage words used	
	SIMSCRIPT program	FORTRAN program
Program storage		
Hand written	9,725	5,072
SIMSCRIPT generated[a]	3,152	...
SIMSCRIPT supplied[b]	4,710	...
FORTRAN supplied	1,566	1,478
IBSYS Library	152	122
IBSYS Nucleus	5,376	5,376
Total program storage	24,681	12,048
Input-output buffers	867	659
Data storage	9,707[c]	20,061

[a] Subprograms generated from the definition cards and the Events List.
[b] Subprograms for the clock, read-in of initial conditions, management of temporary storage, etc.
[c] Includes 2487 words that are reused.

states, etc., at the end of each simulated day, and the end of the run were considered adequate for testing the program.

In the vehicle-dispatching model a number of initial values need to be read in to start a simulation run. For each variable the number of values to be read in for each run can vary. The different kinds of variables used means variations in kinds of numbers and number ranges. If maintenance and cost data as well as run parameters are ignored, it is possible to use up to 3000 vehicles in a run and up to 5000 mission requests in a simulated day. With this kind of variation and quantity, it is desirable to devise a quick and easy way to get the information into the computer. In the vehicle-dispatching model the first thing that was done was to determine that for anticipated uses of the model, the mission requests to be read into the model would be the same for each simulated day of the run. This reduced the maximum number of mission requests for which it might be necessary to read in data from 5000 per simulated day to 5000 per run. The mission request information was also organized so it could be read from 144 cards. The planning for this part of the program also included devising a method for using a variable number to identify the kind of information to read and an end of variable signal to terminate its reading.

Comparison of Storage Used

Table I shows a comparison of the number of words of 7040 core storage used by the SIMSCRIPT and the FORTRAN program for the vehicle-dispatching model. It is immediately evident from this table that the SIMSCRIPT program takes up about twice as much storage space as the FORTRAN program. The storage requirements for the IBSYS Nucleus[1] are the same for the two programs. In all other program categories the SIMSCRIPT program requires more computer storage.

There are several explanations for the SIMSCRIPT storage requirements. One explanation is that the program is made up of many subprograms. The SIMSCRIPT program contains the equivalent of 21 handwritten subprograms (i.e., the SIMSCRIPT Definition Cards and Events List are each counted as a subprogram) compared with 12 handwritten subprograms in the FORTRAN program. These figures do not tell the whole story. As the programs read into the computer, there are about 200 subprograms in the SIMSCRIPT program and about 30 in the FORTRAN program. With the standard subprogram setup required by the IBSYS operating system, 15 words of storage in each subprogram are used to enter and leave each subprogram, and a minimum of 3 words of storage are needed to call for the execution of the subprogram. If each of the SIMSCRIPT subprograms is called for from only one point in the program, well over 3,000 words of storage can be used in subprogram linkage. In many instances SIMSCRIPT reduces to 6 the number of words in a subprogram that are required to enter or leave it. Even with this reduction the SIMSCRIPT program still contains a sizeable block of storage that does nothing but provide operating system compatibility.

A second explanation for the SIMSCRIPT program's use of storage can be found in the inclusion of the program test faclities in the SIMSCRIPT program. Six of the handwritten SIMSCRIPT subprograms are devoted to this. Most of these are SIMSCRIPT reports. The FORTRAN subprograms produced for reports by the SIMSCRIPT Report Generator tend to be long. As a result about 1,800 words of storage in the SIMSCRIPT program are used for program testing. In the FORTRAN program no formal checkout facilities are provided.

Another explanation of the use of storage by the SIMSCRIPT program is the length of the subprograms generated from the handwritten subprograms. The FORTRAN program contains 931 cards. (This excludes comment cards used to explain what is being done in the program.) These cards produce subprograms requiring about 5,000 words of computer storage. In the SIMSCRIPT program 836 noncomment cards produce subprograms requiring almost 10,000 words of storage. In part this comes about, because some SIMSCRIPT statements represent a whole FORTRAN loop and in some instances represent data scanning that requires a loop imbedded in the decision structure of the scan. In addition in some cases the SIMSCRIPT Translator writes FORTRAN programs that require more storage than comparable handwritten FORTRAN programs.

The last reason for the amount of storage used by the SIMSCRIPT program lies in the way computer input and

[1] The IBSYS Nucleus [6] is the part of the IBSYS operating system that stays in core storage while a user's program is being executed. It takes care of the detailed setup necessary for input and output units and calls in the parts of the operating system needed by the next user's job.

output devices are handled. SIMSCRIPT allows the user to specify any FORTRAN input or output unit he wishes. This means that SIMSCRIPT itself always refers to these units with a variable name to which a value is assigned at execution time. This causes FORTRAN to define and supply input/output buffers for all hardware units available. Possible ways of eliminating this in SIMSCRIPT programs are discussed by Weinert and Bossenga [7]. In the two programs being considered, all storage comparisons and timing comparisons are for reading both programs and data from the system input tape and writing output on the system output tape. In Table I the figure for the FORTRAN program shown under input/output buffers includes both the buffers and the file control blocks used by FORTRAN to define the system input unit and the system output unit. FILE cards provided by IBSYS have been used in the SIMSCRIPT program to eliminate the buffers for input/output units that are not used. The different amounts of storage used reflect the different amounts of storage needed for file control blocks to define two units in the FORTRAN program and 13 units in the SIMSCRIPT program.

Table I shows that the amount of computer storage available for data in the SIMSCRIPT program is less than half that available in the FORTRAN program. The amount actually used by the FORTRAN program is about 16,000 words. Data storage limitations on 7040 SIMSCRIPT programs are severe. Ways to deal with the problem both in terms of SIMSCRIPT features and by using machine language programming are discussed by Weinert and Bossenga [7]. In the vehicle-dispatching program no attempt has been made to reduce the storage requirements of the SIMSCRIPT program except for the elimination of unnecessary input/output buffers.

As indicated previously, SIMSCRIPT data storage is divided into two kinds; that used for permanent data and that used for temporary data. The minimum storage requirements for the various types of data in the SIMSCRIPT vehicle-dispatching program are shown in Table II. Assuming that the model is to be executed for three vehicle types, 10 mission-request types, and 30 simulated days, the figures in Table II suggest that, without taking into account the size of the vehicle fleet to be used, about 600 words of computer storage are needed for permanent data. The storage space available for permanent data is that part of the data storage that is not reused. In the vehicle-dispatching program this is a maximum of 7,220 words of storage. Assuming that a fleet of 500 vehicles is to be used, 5,600 words of data storage will be used for permanent data. This will leave about 4,000 words for temporary data. The maximum amount of storage for event notices is required when all vehicles have a vehicle available event scheduled. In this case about 1,500 words of temporary storage will be used for event notices. This does not leave much room for unmet requests at the end of each simulated day; during the day the number of incoming requests must be such that the 500 vehicles available are not swamped.

Now assume that the fleet is to have 1,000 vehicles instead of 500 vehicles. The 10,000 words required for vehicle-data storage alone is more than is available for all the initial conditions. SIMSCRIPT provides for using the Definition Cards that specify model variables to specify data packing. The use of packed data in a SIMSCRIPT program lengthens the subprograms generated by SIMSCRIPT from the Definition Cards and also lengthens

TABLE II. UNPACKED DATA-STORAGE REQUIREMENTS FOR THE SIMSCRIPT VEHICLE-DISPATCHING PROGRAM

Kind of data	Minimum number of words of core storage used
Initial Conditions	
Constants	119
For each vehicle type	41
For each mission request type	4
For each vehicle	10
For each combination of vehicle and mission request type	4 + (4 × NIVT)[a]
For each consecutive combination of vehicle type and scheduled maintenance step[b]	2 + (2 × NIVT)[a]
For each consecutive combination of unscheduled maintenance distribution interval and vehicle type[b]	1 + (NIVT)[a]
For each combination of simulated day and vehicle type	1 + (NIVT)[a]
Temporary data	
For each active mission request	8
For each vehicle-available event notice	3
For other event notices	4

[a] NIVT is number of vehicle types.
[b] This is a two-way table in which the entries in each row are consecutive, but the number of entries in a row need not be the same for every row.

TABLE III. 7040 EXECUTION TIMES[a] FOR SELECTED EXECUTIONS OF THE VEHICLE-DISPATCHING MODEL

Run	Vehicles	Requests /vehicle type /simulated day (approximately)	Simulated days	7040 time, minutes				
				FORTRAN program, vehicle types 1-3	SIMSCRIPT program, vehicle types			
					1-3	1	2	3
Mission Requests Generated								
A	10	10	5	1.5	···	2.4	2.4	2.5
B	25	100	5	1.6	···	3.0	2.9	3.0
C	25	100	15	2.6	···	4.0	4.0	4.1
D	75	100	15	···	8.7	···	···	···
E	100	1000	5	5.2	···	9.1	8.2	7.8
F	200	1600	15	14.0	···	···	···	···
G	600	1600	15	···	···[b]	···	···	···
Mission Requests Read in								
H	25	100	15	1.9	···	···	···	···
I	75	100	15	···	10.2	···	···	···

[a] All times are for programs in relocatable MAP form with both program and data read from the system input tape. IBSYS version 7.1,2 on tape was used for all runs.
[b] The program exceeded its core-storage capacity after processing 1.3 simulated days in 9.2 min.

the execution time of the program. Packed data should therefore be avoided until needed. This is feasible in SIMSCRIPT programs because it can be accomplished by rewriting and retranslating the Definition Cards for the variables that are to be packed. No other program changes are necessary. In the vehicle-dispatching program, if it is assumed that no vehicle will be dispatched on a mission or sent to scheduled or unscheduled maintenance more than 4,095 times, then the vehicle information can be stored in 6 rather than 10 words per vehicle. This will allow the program and the initial conditions for a 1,000 vehicle fleet to read in, but it does not allow for much temporary data storage. At the same time it would be advisable to pack the mission request records into 4 computer words. This cuts the length of each mission request record in half. It should then be possible to use a 1,000 vehicle fleet, if the distribution of the arrival times for incoming requests and the duration of the missions requested are such that when combined with the required maintenance, there is no pile up of unmet requests.

Comparison of Computer Time Used

Many kinds of computer times may be of interest to a computer user. This comparison is limited to two types of 7040 time. One is the 7040 time necessary to obtain from the program as written by the programmer a relocatable deck of cards suitable for reading into the 7040 to make a model run. This will be called program-processing time. The second type of 7040 time to be considered is selected execution times for the model.

The processing time for the SIMSCRIPT program is 16.7 minutes of 7040 time. This is about 2.7 times as long as the 6.2 minutes required for the FORTRAN program. Since 7040 SIMSCRIPT is a FORTRAN pretranslator this time difference is in the expected direction. The amount of the difference may be greater than expected. About one minute of the 16.7 minutes of SIMSCRIPT processing time is used to generate relocatable machine language code from the SIMSCRIPT Definition Cards and Events List. This is usually done only once or at the most only a few times during the development of a program. In addition the FORTRAN code written by the SIMSCRIPT Report Generator tends to be longer than that generated for events or subprograms. This lengthens processing time. During program check out SIMSCRIPT processing is done only on the parts of the program being revised. In SIMSCRIPT programs it is the events and subprograms not the reports that seem to need the most revision. In actual use, over the entire span of program development because SIMSCRIPT is fastest on the parts that need the most revision, the amount of the difference given by these figures can be considered a maximum.

Table III shows some 7040 model execution times using both the FORTRAN program and the SIMSCRIPT program. Each of these runs uses a uniform distribution of 10 mission-request types. Three vehicle types representing relatively short, medium, and long operating and maintenance times are also used in all runs. All runs are also made with only the end-of-run print-out. The selected runs are distinguished by varying the vehicle fleet size, the number of requests to be serviced, the number of days simulated, and the method by which the mission requests are introduced into the model. The variation in vehicle-fleet size and number of requests to be serviced is intended to approximate order-of-magnitude changes in the size of the vehicle-dispatching system represented by the run of the model.

In runs A, B, C, and E the model is operated with only one type of vehicle in the fleet at a time. In all these runs the mission requests are generated by the program. In the FORTRAN program the three vehicle types are processed successively after one read-in of the program and data. In the SIMSCRIPT program, when mission requests are generated the type of vehicle to be used on each request is determined by a random drawing from a distribution of vehicle types. If only one type of vehicle is to be allowed at a time, this means this distribution must be changed for each vehicle type processed. This distribution is read in as part of the SIMSCRIPT initial conditions by SIMSCRIPT routines that are later written over by being added to the SIMSCRIPT temporary data storage. This means it is difficult to change these distributions without reading the program back in with a new set of initial conditions. This is what was done in runs A, B, C, and E. The total time necessary to get answers comparable to those produced from the FORTRAN program is the sum of the times shown for each of the three vehicle types.

Perhaps the most significant thing shown by runs A, B, and C is that until the model runs for 15 simulated days with a fleet of 25 vehicles servicing 100 requests per day, relatively little computer time appears to be used for processing requests. The lack of a significant increase in time between runs A and B indicates that, even when small vehicle fleets are increased by a factor of 2.5 and the number of requests is increased by a factor of 10, most of the computer time is taken up with reading in the programs and data and writing out results.

Run E was originally made with the expectation that a pileup of unmet requests would cause the SIMSCRIPT program to run out of core storage. This does not happen until the even larger vehicle fleet and more disproportionate number of requests shown in runs F and G are used. If the fleet size is such that all the initial conditions can be read in, it is the relations among the model variables for this particular run of the model that determine whether requests are met and also whether the SIMSCRIPT program runs out of storage. In run E, as the model is run with the same fleet size and the same number of requests to be met, the execution time of the SIMSCRIPT program decreases as vehicle-operating and maintenance time, and hence number of unmet requests increases. In other words, so long as the SIMSCRIPT program does not run out of storage and in effect cause execution time to become infinite, it takes less time to recognize that a request cannot be met than to do the data scanning and record keeping that is required to meet it.

TABLE IV. NUMBER OF PROGRAM STATEMENTS CHANGED TO INTRODUCE MODEL CHANGES

Model change	Number of statements changed	
	SIMSCRIPT	FORTRAN
Daily variations of exogenous mission requests	None	69
Multitype vehicle fleet	None[a]	129
Incomplete missions	44	103

[a] This change was included in the original SIMSCRIPT program.

In runs H and I the mission requests are read in rather than generated. The pattern of requests read in is the same as that generated by the FORTRAN program for the first day of run C. This same pattern of requests is used on each simulated day of the run by each vehicle type. The difference in time between runs C and H indicates that for almost identical runs the use of generated missions in the FORTRAN program increases program-execution time by about one-third. This happens because all mission requests are read into the FORTRAN program on a maximum of 144 cards. This set of mission requests is reused for each simulated day with no need to read in additional data. When generated missions are used, a new pattern of requests is generated at the beginning of each simulated day. Runs D and I show that with the SIMSCRIPT program the difference in execution times runs in the opposite direction. When requests are read in the computer, execution time is about one-fifth longer than when they are generated. In the SIMSCRIPT program both generated and read-in requests are assumed to be different each day. In SIMSCRIPT each request read-in is an Exogenous Event that is read-in on a separate card. This means that in run I the SIMSCRIPT program read 4,860 request cards. No advantage was gained from the fact that these cards represented 44 repetitions of the pattern of requests for the first day with variations in day and vehicle type. The same execution time would result if the pattern of daily requests required different mission types, vehicle types, and arrival times each day so long as the total number of missions requested and met remained the same.

Introduction of Model Changes

This comparison will be made on the basis of the number of program statements that need to be changed to incorporate three model changes into both the SIMSCRIPT and the FORTRAN programs for the vehicle-dispatching model. This method of comparison is chosen to make the comparison quantitative. The changes necessary were determined by a programmer reviewing each program and writing out the revisions necessary to incorporate each model change. Since evolutionary model changes often come one at a time, each change is considered independently of the others. Each time the original code was revised to make one change. No computer checkout was made of the revisions. The three model changes considered are (1) the use of arbitrary exogenous mission requests that are different each simulated day, (2) the use of a multitype vehicle fleet, and (3) the introduction of incomplete missions. The results of this comparison are summarized in Table IV.

SIMSCRIPT CHANGES. In the SIMSCRIPT program the changes require an extension of the basic program design decisions. The SIMSCRIPT timing program and the scheduling of external events is set up for models where time is kept track of to 8 significant digits. SIMSCRIPT allows the scheduling of external events at any time. In the vehicle-dispatching model the daily repetition of the pattern of exogenous mission requests was provided by data setup. No special programming was required. As a result the change to completely arbitrary scheduling of external mission requests requires no program change.

In the SIMSCRIPT program once it was decided to accumulate summary information separately by both vehicle and mission-request type the use of a multitype vehicle fleet was an obvious extension of the model. To accomplish this model extension, each mission request whether generated or read in needs to carry with it the type of vehicle to be used. When a vehicle is available to meet a request it is necessary to determine if it is the type needed. This changes the kind of delayed mission search used. It requires the removal of the request for the available type of vehicle. This meant changes in 18 program statements. This was easy; so it was included in the original SIMSCRIPT program.

The introduction of incomplete missions requires the addition of some new variables. New random variables are needed to represent the amount of each mission that is completed and the effect of incomplete missions on unscheduled maintenance. Another variable is needed to indicate how much of a mission needs to be completed before it need not be rescheduled. A new incomplete mission event that takes place when activity on the incomplete mission ceases must be added to the Events List, and event notices, and the event must be coded. Cumulative attributes need to be adjusted for the occurrence of the incomplete mission. These changes are additions to the program. They are added at the appropriate points to the existing program structure without changing that structure.

FORTRAN CHANGES. In the FORTRAN program each of these changes requires a review of the basic program design decisions. For arbitrary daily patterns of exogenous mission requests it is the reading of initial input data that needs to be reconsidered. Data must now be read in both at the beginning of a run and sometime during each simulated day. The revision worked out removed the mission requests from the initial run data, but retained the variable number for identifying the data and the end-of-variable signal. The revised program reads in external mission requests at the beginning of each simulated day. This means that if three vehicle types are being used the mission requests for the whole run will be repeated three times. This approach simplified the coding changes and meant the addition of

not more than 288 cards per simulated day for each vehicle type.

The introduction of a mixed vehicle fleet into the FORTRAN program resulted in a review of the mission request table. The revision decided on changed the meaning of entries in this table. The values entered were recoded to represent the combination of mission type and vehicle type requested. To minimize the coding changes required, the amount of storage allocated to keeping mission type and vehicle type information was left unchanged. This means that the maximum number of different kinds of missions to which a vehicle can be assigned is reduced from 20 to 6. The maximum number of vehicles in the fleet is still 1,000. For a run the maximum number of vehicles of one type is 334.

The inclusion of incomplete missions in the FORTRAN program required another review of the setup of the mission-request table. Since these changes were made independently, it was decided to leave the table set-up as originally designed in order to retain the fast execution times that come from reading in a minimum amount of data. This meant it was necessary to introduce a separate table of incomplete missions containing the mission type and the time period in which it is to be rescheduled. When this table is set up, a reasonable amount of storage can be allocated to it, but it is wise to have the code check to see if the table capacity is exceeded and if necessary provide for terminating program execution. The introduction of this table also requires changes in scanning the mission request table to include scanning the incomplete mission table. At the end of each simulated day the incomplete mission table needs to be included in the tally of missions not completed and the remaining missions cancelled.

Implications

The use of SIMSCRIPT and FORTRAN on the vehicle-dispatching model illustrates situations that can recur on many small to medium sized simulation problems. First, let's assume that there is someone available to work on the problem who understands SIMSCRIPT and sees no problem features that are obviously troublesome to handle. Then the use of SIMSCRIPT allows the computer program preparation to take place from the point of view of the model being simulated. In contrast, the use of an algebraic language causes the problem to be approached from the point of view of the computer. Things like coding ease and computer storage allocation that are extraneous to the model design dominate the program choices. With an algebraic language programming time is spent providing for this problem in particular facilities that are already available in SIMSCRIPT in a generalized form. In the vehicle-dispatching model this is true for representing the passage of simulated time, for allocating computer storage, and for providing initial data read in. The need for these facilities is typical of simulation problems so the use of an algebraic language on simulation problems means that there will be many particular schemes devised to provide these facilities. If competent programmers devise these schemes, the advantages of fast execution times and parsimonious use of storage as illustrated by the vehicle-dispatching model can result.

The introduction of model changes into the vehicle-dispatching model programs also illustrates a recurring condition in simulation problems. The development of a simulation model is often an evolutionary process. Sometimes it is only after a model has been used that desirable model features are recognized. In simulation problems it also happens that a new model slightly different from a previous model is needed. In both cases it can be a material advantage to be able to put model changes into a simulation program easily. SIMSCRIPT features tend to be generalized versions of what is needed in many different simulation models. These generalized features have been provided by a series of open ended program structures built up of many short subprograms. This kind of program structure usually makes the introduction of model changes relatively easy. As an indication of this in the vehicle-dispatching model it takes almost 7 times as many changes in the FORTRAN program to provide the model changes considered as it does in the SIMSCRIPT program.

Now let's assume there is no one available who knows how to use SIMSCRIPT. At the present time there are many more programmers who know how to use an algebraic computer language such as FORTRAN than know how to use SIMSCRIPT. In order to take advantage of SIMSCRIPT features, how much time must be spent learning it? The speed with which SIMSCRIPT is learned is like many other learning tasks—it is subject to wide individual differences. It is possible for an experienced programmer to learn SIMSCRIPT with no formal instruction. There is enough written material available that it is even possible to do this when it is necessary to set up the SIMSCRIPT Translator and the SIMSCRIPT System Routines before using them. A better feel for expected performance may be obtained by examining the results of two similarly conducted SIMSCRIPT classes. Both classes provided three-hour class sessions on five consecutive days. In each case the instruction was organized around a class problem. Each member of the class was assigned at least one SIMSCRIPT event, report, or subprogram. An error-free SIMSCRIPT translation of this assignment was considered evidence of successful completion of the course. In one class [8] 19 people attended one or more class sessions. Ten of these people attended all the class sessions and eight of them produced error-free SIMSCRIPT translations in not more than four computer runs. In this same class one experienced COBOL programmer dropped out because he couldn't understand SIMSCRIPT. At the same time one successful member of the class had only a limited amount of assembly language experience on a different kind of computer. With minor assistance he checked out the entire program and used it to make the desired simulation runs. In the second class there were 17 people. Eight of these attended all sessions, but only three were successful. The interest that arises when people recognize SIMSCRIPT features that make it easy to describe their problems probably makes learning

easier. This kind of difference in motivation as well as differences in ability and experience could account for wide variability in learning speed.

The amount of computer time used by the two vehicle-dispatching model programs will not be the same for other problems, but longer SIMSCRIPT times can be expected because they illustrate the effect of certain language implementation decisions. 7040 SIMSCRIPT like the original 7090 SIMSCRIPT is in part a FORTRAN pretranslator. This means that SIMSCRIPT processing times will be longer because they will include FORTRAN processing time. As the vehicle-dispatching problem illustrates this time can be almost 3 times as long. The pretranslator aspect of SIMSCRIPT also contributes to its longer execution times. This same pretranslator structure, however, has advantages. It together with the existence of 7090 SIMSCRIPT meant that only 8 man-months of effort were necessary to construct 7040 SIMSCRIPT. Since 7040 SIMSCRIPT is a FORTRAN pretranslator, SIMSCRIPT deficiencies can be overcome by using FORTRAN for parts of the program. Without the pretranslator structure the language deficiencies can be alleviated by writing part of the program in assembly language code. This may be quite satisfactory for a full time programmer, but is a stumbling block for the researcher with a simulation problem who is using a simulation language to make the computer more responsive to his needs. There have been suggestions [9] for making the simulation language more responsive by giving the user the ability to add his own language features. This in turn means a more difficult language construction job and a more sophisticated user. Whether the computer time used by a simulation language [5, 10], is a desirable investment in any particular problem depends upon the resources available for the problem, and the trade-offs that can be made among them.

The increased storage requirements of SIMSCRIPT illustrated by the vehicle-dispatching model are substantial. This characteristic of simulation languages intensifies the storage shortage found in many simulation problems. The storage shortage can be acute. Take for example a problem that uses particle velocities at over 350,000 points as one variable. Without considering any other data or any program storage requirements, the values of this variable would have to be packed more than 10 to a word to fit into the core storage of a 36-bit word computer with 32K words of core storage. SIMSCRIPT attempts to deal with this problem by providing reuse of temporary entity storage and postponing storage decisions until execution time to take advantage of specific run trade offs. In some problems this is satisfactory, but inadequate storage is a continuing problem for simulation language, operating system, and computer designers. 7040 SIMSCRIPT illustrates how language and operating system design decisions both contribute to the storage shortage. SIMSCRIPT achieves its run time data storage allocation by using short subprograms with entry points for getting and storing attribute values. This produces a program with many subprograms that will be referred to by other subprograms. The 7040 IBSYS operating system was designed with a minimum 18-word standard subprogram linkage. It was also designed with a program loader that limited the number of inter-subprogram references. Both of these operating system design features limit SIMSCRIPT potentialities. In the early distributed versions of IBSYS the inter-subprogram reference limit was so low that it was impossible to load a nontrivial SIMSCRIPT problem. In a program with many subprograms the lengthy IBSYS subprogram linkage uses valuable data storage that can be only partially compensated for by the program chaining facilities provided.

This comparison of a SIMSCRIPT program and an algebraic computer language program for a vehicle-dispatching model has illustrated the effect of SIMSCRIPT on program design decisions and resulting program characteristics. The algebraic language can provide a program that is more parsimonious in its use of computer storage and time. SIMSCRIPT provides program design tied to model characteristics and produces a program into which it is easy to introduce model changes. To use SIMSCRIPT it may be necessary to train someone in its use, and provide more computer time and storage. For small or moderate size simulation problems these requirements appear to be manageable. For large simulation problems storage requirements will need the continued attention of simulation language, operating system, and computer designers.

RECEIVED DECEMBER 1966; REVISED AUGUST, 1967

REFERENCES

1. TEICHROEW, D., AND LUBIN, J. F. Computer simulation—discussion of the technique and comparison of languages. *Comm. ACM 9*, 10 (Oct. 1966), 723–741.
2. TOCHER, K. D. Review of simulation languages. *Oper. Res. Quart. 16*, 2 (June 1965), 189–217.
3. KRASNOW, H. S., AND MERIKALLIO, R. A. The past, present and future of general simulation languages. TM 17-7004, IBM Advanced Sys. Develop. Div., Aug. 1963.
4. YOUNG, K. A user's experience with three simulation languages (GPSS, SIMSCRIPT & SIMPAC). TM-1755/000/00, System Development Corp., Santa Monica, Calif., Feb. 1964.
5. EDWARDS, P. G., AND MURPHY, J. G. A comparison of the use of the GPSS and SIMSCRIPT simulation languages in designing communications networks. TM-03969, Mitre Corp., Bedford, Mass., Mar. 1964.
6. IBM 7040/7044 operating system (16/32K) programmer's guide. Form C28-6318-4, IBM Corp., Oct. 1964.
7. WEINERT, A. E., AND BOSSENGA, J. R. 7040 SIMSCRIPT: a case study. RAC-TP-196, Research Analysis Corp., McLean, Va., Feb. 1966.
8. WEINERT, A. E. Learning a simulation language. Proceedings of the ARO-Working Group on Computers, ARO-D Report 65-1, U.S. Army Research Office—Durham, Durham, N.C., Feb. 1965, pp. 350–365.
9. KIVIAT, P. J. Introduction to SIMSCRIPT II programming language. P-3314, RAND Corp., Santa Monica, Calif., Feb. 1966.
10. ROSEN, S. B., MCCABE, J. P., NEVANS, E. S., WALDEN, R., FALVEY, J., AND PACCHIOLI, A. Simulation and analysis of 473L system. Vol. II. SIMCOM and SIMSCRIPT simulation techniques; a comparison. ESD-TDR-64-656, General Electric Co., Washington, D.C., Dec. 1964.