SIMULATION MODELING WORKSHOP

William E. Biles

ABSTRACT

This paper reviews the fundamentals of computer simulation modeling. Simulation is viewed here as a technique for the experimental manipulation of a model of a real-world system, drawing heavily upon computer science, mathematics, probability and statistics. Simulation involves modeling the entities in a physical system and the activities in which those entities engage. It affords the opportunity to construct ideally configured systems and select optimum system operating conditions on the basis of results obtained by simulating such configurations and conditions on a digital computer. Models of some simple material handling systems are used to illustrate the simulation concepts presented here.

INTRODUCTION

Simulation is a problem solving procedure for defining and analyzing a model of a system. Simulation can take several forms, including electrical analog, fluid analog, and the more familiar digital computer simulation. This paper focuses on the latter. In that context, simulation can be defined as the establishment of a mathematical-logical model of a system and the experimental manipulation of that model on a digital computer.

This paper provides a survey of computer simulation. It describes a frame of reference by which the salient features and characteristics of a system are captured in a computer model. It briefly reviews the basic steps in model development and operation, and surveys various simulation languages which can be employed in structuring the model. This paper does not purport to give a detailed treatment of computer simulation, but rather attempts to motivate practitioners to take advantage of a powerful, readily available, and relatively inexpensive analytical tool.

Computer simulation offers a convenient means of studying the behavior of a system. By system, we mean some circumscribed sector of reality upon which we focus analysis for the purpose of accomplishing some logical end. A system is a collection of related entities, each characterized by attributes. These entities engage in activities, which elapse over time and culminate in events. An activity may last some known or deterministic time, or some uncertain or probabilistic time. An event, which marks the termination of an activity, is generally such that it alters the state of the system by changing the values of the attributes associated with one or more of its entities.

To illustrate these basic concepts, consider a simple material handling system consisting of the following entities and their associated attributes:

Entity	Attribute
1. Fork/truck driver	Status (idle, moving loaded, moving unloaded)
2. Receiving dock	Number of pallet loads of received goods waiting for placement in storage
3. Pallet load	Class of goods (slow mover, moderate mover, fast mover)
4. Rack storage slot	Status (empty, occupied)

Activities in which the fork truck/driver entity can engage are (1) waiting at the receiving dock, (2) moving to a rack storage slot with a pallet load, and (3) moving to the receiving dock empty. The events corresponding to the end of these activities are, respectively: (1) arrival of a pallet of received goods on the receiving dock, thus ending the "waiting" activity in which the fork truck/driver entity has possibly been engaged; (2) arrival of the loaded form truck/driver entity at a designated rack storage slot; and (3) arrival of the empty fork truck/driver entity at the receiving dock. The first of these events alters the state of the receiving dock by adding one pallet, and if the dock has been empty and the fork truck/driver entity idle, changes the attribute of the fork truck/driver entity by making it "move loaded". The second event alters the state of rack storage slot from "empty" to "occupied", and changes the status of the fork truck/driver to "moving empty". The third event changes the status of the fork truck/driver and possibly the dock status as well. Thus, the operation of this system involves a number of interrelated entities, attributes, activities, events and system states changing values over time.

Simulation is the action of performing experimentation on a model of a given system. A model is a representation of a system based on theory, empirical observation, or a combination of both. Thus a computer simulation model of a system must represent

the system entities, maintain values for the attributes to these entities, cause the entities to engage in activities, and mark the occurrence of events at the culmination of these activities. The simulation model must be able to represent the passage of time, expanding or compressing time as necessary to achieve the desired experimentation.

An important consideration in any computer simulation is the necessity to identify and control sources of variation, eliminating unwanted variation in order to assess the relationship between independent (input) variables and dependent (output) variables. Moreover, it is occasionally desirable, in the course of a simulation, to stop the experiment and review the results to date. This means that all phenomena associated with the experiment must retain their current values or states until the experiment resumes. Finally, it is necessary at the conclusion of a simulation experiment to perform an analysis of the results. In many cases it is desirable to replicate the experiment; that is, repeat the simulation at the same conditions to examine the effects or random variation on the results.

Computer simulation allows us to do all these things. It provides the ability to experiment, to test and evaluate proposed systems or proposed changes in existing systems in advance of investing time, money and effort in their physical development. Computer simulation affords the opportunity to evaluate proposed system configurations and operating policies without interfering with the operation of any existing system or constructing a new system.

This paper presents the basic approach to developing and operating a simulation model of a real system, reviews several languages one might apply in develop- a simulation model, and suggests procedures one might apply in performing an analysis of the input parameters and output variables.

MODEL DEVELOPMENT

In computer simulation, we attempt to develop a model which captures the important features of the system under study. One approach to describing the basic features of a given system is to adopt the "black box" view as illustrated in Figure 1. The purpose of any systems analysis is to optimize one or more measures of effectiveness. For the illustrative problem described earlier, a suitable measure of effectiveness might be the time a pallet of received goods remains on the dock. For example, if the goods are frozen foods and the receiving dock is maintained at ambient temperature, it is desirable to minimize dock "residence" time. A decision variable might be the number of trucks assigned to the bin stocking operation, so that appropriate values for this variable could be 1,2,3,... An uncontrollable variable might be dock temperature, depending as it does on climatic conditions. (Of course, in a variant of this problem, the dock temperature could be brought under control of the decision-maker by installing refrigeration units, at some investment of capital.) Thus, in the "black box" approach to model development, the simulationist must (1) identify the measures of system effectiveness, (2) establish those factors which can be controlled in the design and operation of the system, and (3) identify, if possible, those variables which cannot be controlled. This is the first step in developing a credible model of the system.

The second step is to formulate the logical interactions among the entities represented in the model, to provide a means of generating random variations in the several variables in the model, to cause time to flow from the start to the termination of a simulated time period, to collect statistical data on the model variables over the course of the simulation, and to generate a report at the conclusion of the experiment. These procedures are performed by constructing a flow chart which represents the mathematical and logical operations embedded in the model. This flow chart is then translated into a computer program, using the selected language. This program must be verified, which means that one must establish that the program actually possesses the features intended for the model. Verification is an essential step in the development of a credible model of the system.

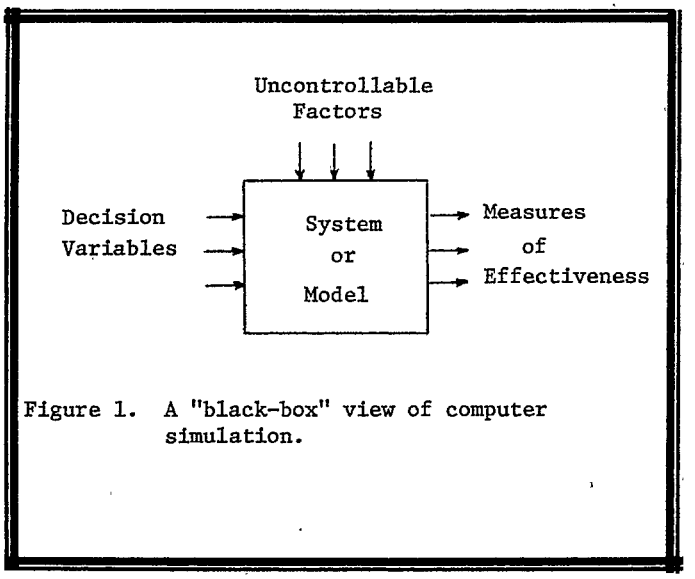


Figure 1. A "black-box" view of computer simulation.

The final step in the simulation process is the experimental manipulation of the computer model. This involves assigning values to the decision variables, executing the model at these conditions, and analyzing the results of the simulation. When the values of the decision variables represent conditions which have been observed for the real system, one is able to validate the model. Validation is the process by which one establishes that the model behaves like the real system, and is a crucial step in producing a credible model. Thus the credibility of a model is ascertained through the joint processes of verification and validation.

SIMULATION LANGUAGES

One of the first steps in model development is the

choice of a simulation language. Simulation models of real-world systems can often be programmed in one of the problem-oriented languages such as FORTRAN-IV, ALGOL AND PL/1. However, these languages require extensive programming to establish data structures, define the classes of entities within a system, adjust the numbers and attributes of such entities as time flows, schedule events and cause their occurrence within the flow of time, maintain statistics during the progress of the experiment, and generate the detailed summary reports needed at the termination of the experiment. Schmidt and Taylor [11] provide an excellent treatment on the use of FORTRAN to develop simulation models.

Several programming languages exist for the express purpose of simulation, and these typically require far less programming than the general purpose languages. These simulation languages are classified as either discrete-event or continuous, depending on the manner in which the flow of time is treated. Discrete-event simulation languages, including GPSS [12], SIMSCRIPT-II [4], SIMULA [1] and GASP-II [7], advance time either by moving in a discrete increment Δt or by moving directly from one event to the time of the next scheduled event. The continuous simulation languages such as CSMP [3], MIMIC [5], AND CSSL [13] are typically applied to those time-dependent processes which are modeled through differential equations. Time is advanced in very small increments, approximating the dt interval in the derivative dx/dt. Two languages, GASP-IV, a FORTRAN-based language [8], and GASP-PL/1, a PL/1-based language [9] afford both discrete-event and continuous simulation modeling.

There are other simulation programs which exist for very specialized applications. One which could be especially useful in modeling material handling systems in GERTS [15]. This program models a system as a stochastic network, with nodes representing events and branches representing the passage of time between events. (The term "stochastic" means probabilistic with respect to the variable "time".) GERTS requires no programming, but employs a data set which describes the characteristics of the nodes and branches in a network representation of the real system. A recent development called QGERTS [10] enables the modeling of queues, a very important aspect of material handling systems.

The choice of a particular simulation language will often depend on the type of computer hardware one has available, but it is important to choose a language which will have wide applicability within the technology of one's firm or agency.

MODEL OPERATION

There are several processes involved in the operation of a discrete-event probabilistic simulation model. Two which are especially important are random number generation and stochastic variate generation, the former usually a component of the latter. To illustrate these two processes, let us focus attention on one aspect of the simple material handling system described earlier.

Suppose that there are three classes of goods, fast-moving items, moderate-moving items and slow-moving items, based on annual sales volume. For the sake of illustration, suppose that 10% of the pallets received are fast-movers, 60% are moderate-movers and 30% are slow-movers. Stating these as probabilities we get P(F) = 0.1, P(M) = 0.6 and P(S) = 0.3, respectively. In simulating the arrival of a pallet of goods at the receiving dock, we might wish to randomly generate the movement class (F,M or S) of that particular pallet. We could assign the numbers 00 to 09 to fast movers, 10 to 69 to moderate movers, and 70 to 99 to slow movers, and use a table of uniformly distributed two-digit random numbers to generate the movement class of any given pallet. For example, suppose that we want to generate the classification of twenty newly arrived pallets of goods. Suppose we arbitrarily choose a column in the table which yields the following sequence of two-digit random numbers:
84,07,55,93,30,10,43,21,67,28,05,10,42,43,88,
50,08,52,12,36
The corresponding classes of item movement are as follows:
S,F,M,S,M,M,M,M,M,M,F,M,M,M,S,M,F,M,M,M.
Thus,in simulating the arrival of twenty pallets,we have generated 3 fast movers (15%), 14 moderate movers (70%), and 3 slow movers (15%). If we were to simulate a large number of arrivals, say 10000, we would expect to more closely approach to expected proportions than we have in this very small simulation.

Observe the two processes which we have employed here. We first selected a set of 20 two-digit random numbers from a table and then used a specific random number (say 30) to generate a specific value of a random variable (M). The first of these steps constitutes random number generation, the second random variate generation. We have done this without the use of a computer, but similar processes are executed when we employ the computer.

Before examining how the computer would be made to generate random numbers, it is instructive to examine the properties of random numbers. As was stated earlier, the table of two-digit random numbers to which we referred was a table of uniformly distributed random numbers. That is, each of the 100 numbers in the interval (00, 99) has an equal chance of being selected, so that if we let Y represent the random number, then

$$f(Y) = \frac{1}{100} \qquad 00 \leq Y \leq 99$$

Three considerations play influential roles in determining whether or not a particular set of random numbers are adequate for the purpose of simulation. The numbers must pass a battery of statistical tests designed to reveal departures from independence and uniformity. For a truly random sequence $Y_1, Y_2, \ldots$, the elements of any subsequence of these numbers must be jointly independent and each Y_i in the sequence must come from a uniform distribution. Failure of a sequence of random numbers to possess these properties can lead to severely misleading results in simulation work.

Schmidt and Taylor [11], Shannon [14], and Fishman [2] provide excellent treatments of the statistical procedures used for testing random number sequences. Phillips [6] presents a computer program which automatically applies any one of four statistical

tests to a set of observations for each of several probability distributions including the uniform distribution. Phillips' goodness of fit package, or one of equivalent capability, is practically indispensible to sound simulation work.

For the moment let us assume that the random number generator program does produce acceptable numbers for our application. There are other desirable properties which the program should have. These are as follows:

1. It should be fast. That is, it should generate a random number in the minimum amount of time in order to keep simulation running time as low as possible.

2. The program should be short, so as to minimize the core storage for the program itself.

3. It should have a long period. That is, it should generate a long sequence of random numbers before the same sequence reappears.

4. The generator should be able to reproduce the same sequence of numbers at will, so that we can duplicate the experiment as needed.

5. The generator should not degenerate. Degeneracy is the condition in which the same number is continuously reproduced by the generator.

6. The generator should be algorithmic, so that the i-th term in the sequence is used to calculate the (i+1)-st term, the (i+1)-st term the (i+2)-nd term, and so on.

There are several methods by which random numbers are generated. The earliest method was the mid-square technique, in which each successive number is generated by taking the middle n-digits of the square of the previous n-digit number. A similar technique is the mid-product technique, where the middle n-digits of the current number is multiplied by the middle n-digits of the previous number.

The random number generators in common use now are based on congruential methods. The multiplicative congruential method is one in which successive numbers are related by

$$Y_{i+1} = aY_i \pmod{m}$$

This means that the product aY_i is divided by m and the remainder is called Y_{i+1}. The values of a and m must be chosen in light of the number of binary bits used to form an integer word in a given computer. For example.

$$Y_{i+1} = (2^{17} + 3)\, Y_i \pmod{2^{35}}$$

has been shown to produce acceptable sequences of random numbers on machines having 35-bit integer words.

The process of producing random variates uses the uniformly distributed random numbers $0 \leq R \leq 1$, and makes use of the relationship between a random variable X and its cumulative distribution function F(X). Since from the laws of probability $0 < F(X) < 1$ the range of values for the cumulative distribution function happens to coincide with that for R. Hence, one must simply invoke the relationship.

$$X = \phi[F(X)]$$

where $\phi\,[F(X)]$ is the inverse of the cumulative distribution function F(X). This relationship is illustrated in Figure 2 (a) and (b).

For example, suppose that pallets arrive on the receiving dock according to a Poisson distribution with mean rate μ pallets per minute. Then the random variable X, the time between arrivals, follows an exponential distribution with parameter $\lambda = 1/\mu$, so that the probability density function is

$$f(x) = \lambda e^{-\lambda x}, \quad 0 \leq x \leq \infty$$

and the cumulative distribution function is

$$F(x) = 1 - e^{-\lambda x}$$

The parameter λ is the mean time between arrivals. Solving for x in terms of F(x) we get

$$e^{-\lambda x} = 1 - F(x)$$

Letting $R = F(x)$, we have

$$e^{-\lambda x} = 1 - R$$

Taking the natural logarithm of each side, we obtain

$$-\lambda x = \ln\,(1-R)$$

so that

$$x = -\frac{1}{\lambda} \ln\,(1-R)$$

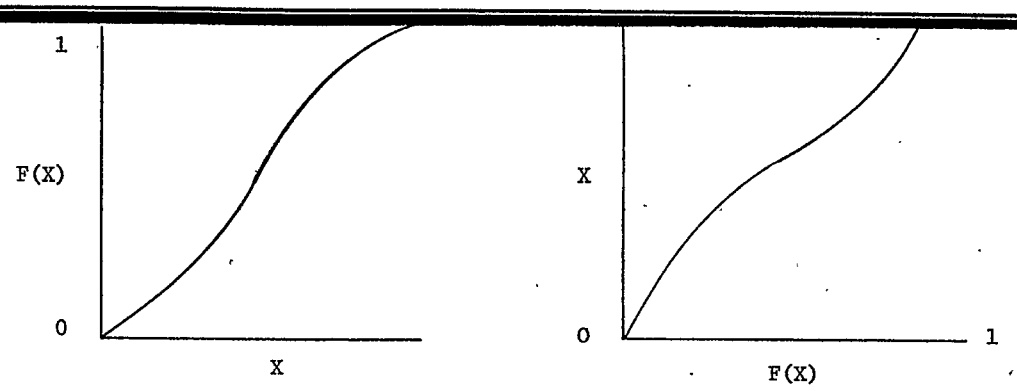
Thus to generate an exponentially distributed random variate x we first generate a random number R between 0 and 1 and solve for x. For example if the mean arrival rate of pallets is 0.4 per minute, the mean time between arrivals is 1/0.4, or 2.5 minutes. Suppose we generate the random number 0.56. Then the randomly generated time between arrivals is

$$x = -\frac{1}{2.5} \ln\,(1-0.56) = -0.4 \ln\,(0.44) = (-0.4)(-0.821)$$

$$x = 0.328 \text{ minutes}$$

Similar random variate generators can be constructed for any number of discrete and continuous probability distributions, or even for empirical distributions.

We have seen how the computer can be made to generate the random numbers and random variates needed for a credible simulation model. These tools can be used to generate many random variables in a simulator, but one of the most important functions is the generation of events. In the above example, if we simply start a simulation with the arrival of a pallet of goods, we can use the occasion of



(a) Cumulative Distribution Function. (b) Inverse CDF

Figure 2. Constructing a Random Variate Generator.

that arrival to generate the next arrival, and so on. Likewise, we have seen how we can use random number generation to assign a particular characteristic, such as class of item movement, to the newly arrived pallet. By creating a program which causes several types of events, and carefully establishes the logic for the occurrence of each event of the same type, we can proceed from some starting time to some completion time. In this fashion, we can simulate work shifts, weeks, or years of elapsed time. We can simulate queues, inventory systems, machining operations, court trials, surgeries, and a myriad of other activities in which men, materials and machines engage. It should not be difficult to visualize how we can employ simulation to analyze many different facets of real-world systems.

REFERENCES

1. Dahl, O., and K. Nygaard, "SIMULA-an ALGOL-Based Simulation Language," Communications of the Association of Computing Machinery, Vol. 9, No. 9, September 1966, pp. 671-678.

2. Fishman, G. S., Concepts and Methods in Discrete Event Digital Simulation, John Wiley and Company, New York (1973).

3. IBM Corporation, "Introduction to 1130 Systems Modeling Program II," GH20-0848-1, White Plains, N.Y. (1970).

4. Kiviat, P. J., R. Villanueva, and H. Markowitz, "The SIMSCRIPT-II Programming Language," Prentice-Hall, Englewood Cliffs, N.J. (1969).

5. Petersen, N. D., "MIMIC, An Alternative Programming Language for Industrial Dynamics," Journal of Socio-Economic Planning Science, Vol. 6, June 1972, pp. 319-327.

6. Phillips, D. T., "Applied Goodness of Fit Testing," AIIE Monograph Series, AIIE-OR-72-1, Atlanta, Georgia (1972).

7. Pritsker, A.A.B., and P. J. Kiviat, Simulation with GASP-II, Prentice-Hall, Englewood Cliffs, N.J. (1969).

8. Pritsker, A.A.B., "The GASP-IV Simulation Language,Wiley-Interscience, New York (1974).

9. Pritsker, A.A.B., and R. E. Young, Simulation with GASP-PL/1, John Wiley and Sons, New York (1975).

10. Pritsker, A.A.B., The Q-GERT User's Manual, Pritsker and Associates, Inc., 1201 Wiley Drive, West Lafayette, Ind., September 1973.

11. Schmidt, J. W., and R. E. Taylor, Simulation and Analysis of Industrial Systems, Richard D. Irwin, Inc. Homewood, Illinois. (1970).

12. Schriber, T., A GPSS Primer, Ulrich's Books, Inc., Ann Arbor, Michigan (1972).

13. SCi Simulation Software Committee, "The SCI Continuous System Simulation Language (CSSL)," SIMULATION, Vol. 9, December 1967, pp. 281-303.

14. Shannon, R. E., Systems Simulation: The Art and Science, Prentice-Hall, Inc., Englewood Cliffs, N.J. (1975).

15. Whitehouse, G. E., Systems Analysis and Design Using Network Techniques, Prentice-Hall, Inc., Englewood Cliffs, N.J. (1973).