Introduction to Operations Research
Sixth Edition 1995
Frederick S. Hillier
Professor of Operations Research
Stanford University
Gerald J. Lieberman
Professor Emeritus of Operations Research
and Statistics
Stanford University
OFERTA
DA EDITORA McGRAW-HILL DE PORTUGAL, Lda.
McGraw-Hill, Inc.
New York St. Louis San Francisco Auckland Bogotá Co
Lisbon London Madrid Mexico City Milan Montreal
New Delhi São Paulo Singapore Sydney Tokyo Toronto

model rather than on
venient, expensive, ar
viewed as virtually indis
should be based upon sound statistical theory.

This chapter focuses on *discrete event simulations* (as opposed to *continuous simulations*), i.e., simulations where changes in the *state of the system* occur at random points in time (as opposed to continuously) as a result of the occurrence of discrete events. The basic building blocks of a model for a discrete event simulation are the possible states and events, a simulation clock for recording the passage of (simulated) time, a mechanism for randomly generating the different kinds of events, and a mechanism for then generating state transitions.

These building blocks are described in Sec. 21.1 in the context of illustrative examples. The second section elaborates on the formulation and implementation of simulation models. The next two sections then focus on the design and analysis of the program of statistical experimentation inherent in a simulation study.

21.1 Illustrative Examples

In this section we use some relatively simple stochastic systems to introduce and illustrate some basic concepts of simulation. The first system is so simple, in fact, that the simulation does not even need to be performed on a computer. The second system incorporates more of the normal features of a simulation, although it, too, is simple enough to be solved analytically. Following these two examples, we survey some more typical applications of simulation.

Example 1: A Coin-Flipping Game

Suppose you are offered a chance to play a game in which you repeatedly flip an unbiased coin until the *difference* between the number of heads tossed and the number of tails tossed is three. You are required to pay \$1 for each flip of the coin, but you receive \$8 at the end of each play of the game. You are not allowed to quit during a play of the game. Thus you win money if the number of flips required is fewer than eight, but you lose money if more than eight flips are required. How would you decide whether to play this game?

Many people would base this decision on simulation, although they probably would not call it by that name. (There is also an analytical solution for this game, but it is not a particularly elementary one.) In this case, simulation amounts to nothing more than playing the game alone many times until it becomes clear whether it is worthwhile to play for money. Half an hour spent in repeatedly flipping a coin and recording the earnings or losses that would have resulted might be sufficient. This is a true simulation because you are *imitating* the actual play of the game *without* actually winning or losing any money.

How would this simulated experiment be executed on a computer? Although the computer cannot flip coins, it can generate numbers. Therefore, it would generate (or be given) a sequence of random digits, each corresponding to a flip of a coin. (The generation of random numbers is discussed in Sec. 21.2.) The probability distribution for the outcome of a flip is that the probability of a head is $\frac{1}{2}$ and the probability of a

tail is $\frac{1}{2}$, whereas there are 10 possible values of a random digit, each having a probability of $\frac{1}{10}$. Therefore, five of these values (say, 0, 1, 2, 3, 4) would be assigned an association with a head and the other five (say, 5, 6, 7, 8, 9) with a tail. Thus the computer would simulate the playing of the game by examining each new random digit generated and labeling it a head or a tail, according to its value. It would continue doing this, recording the outcome of each simulated play of the game, as long as desired.

To illustrate the computer approach to this simulated experiment, we suppose that the computer generated the following sequence of random digits:

8, 1, 3, 7, 2, 7, 1, 6, 5, 5, 7, 9, 0, 0, 3, 4, 3, 5, 6, 8, 5,
8, 9, 4, 8, 0, 4, 8, 6, 5, 3, 5, 9, 2, 5, 7, 9, 7, 2, 9, 3, 9,
8, 5, 8, 9, 2, 5, 7, 6, 9, 7, 6, 0, 7, 3, 9, 8, 2, 7, 1, 0, 3,
2, 6, 2, 7, 1, 3, 7, 0, 4, 4, 1, 8, 3, 2, 1, 3, 9, 5, 9, 0, 5,
0, 3, 8, 7, 8, 9, 5, 4, 0, 8, 3, 8, 0, 1.

Thus, if we denote a head by H and a tail by T, the first simulated play of the game is THHTHTHTTTT, requiring 11 simulated flips of a coin. The subsequent simulated plays of the game require 5, 5, 9, 7, 7, 5, 3, 17, 5, 5, 3, 9, and 7 simulated flips, respectively. This experiment has a sample size of 14 (14 simulated plays of the game), where the individual observations are the number of flips required for a play of the game. One useful statistic is

$$\text{Sample average} = \frac{11 + 5 + \cdots + 7}{14} = 7,$$

because the sample average provides an *estimate* of the true *mean* of the underlying probability distribution.

This sample average of 7 would seem to indicate that, on the average, you should win about \$1 each time you play the game. Therefore, if you do not have a relatively high aversion to risk, it appears that you should choose to play this game, preferably a large number of times.

However, *beware*! One common error in the use of simulation is that conclusions are based on overly small samples, because statistical analysis was inadequate or totally lacking. In this case, the *sample standard deviation* is 3.67, so that the estimated *standard deviation* of the *sample average* is $3.67/\sqrt{14} \approx 0.98$. Therefore, even if it is assumed that the probability distribution of the number of flips required for a play of the game is a *normal distribution* (which is a gross assumption because the true distribution is *skewed*), any reasonable *confidence interval* for the true *mean* of this distribution would extend far above 8. Hence a much larger sample size is required before we can draw a valid conclusion at a reasonable level of statistical significance. Unfortunately, because the standard deviation of a sample average is inversely proportional to the *square root* of the sample size, a large increase in the sample size is required to yield a relatively small increase in the precision of the estimate of the true mean. In this case, it appears that an additional 100 simulated plays of the game might be adequate.

It so happens that the true *mean* of the number of flips required for a play of this game is 9. Thus, in the long run, you actually would lose about \$1 each time you played the game.

Although formally constructing a full-fledged *simulation model* is not really necessary for this simple simulation, we do so now for illustrative purposes. The *stochastic system* being simulated is the successive flipping of the coin for a play of the

game. The *simulation clock* records the number of (simulated) flips t that have occurred so far. The information about the system that defines its current status, i.e., the *state of the system*, is

$$N(t) = \text{number of heads minus number of tails after } t \text{ flips.}$$

The *events* that change the state of the system are the flipping of a head or the flipping of a tail. The *event generation mechanism* is the generation of a *random digit* where

$$0 \text{ to } 4 \Longrightarrow \text{a head,}$$
$$5 \text{ to } 9 \Longrightarrow \text{a tail.}$$

The *state transition mechanism* is to set

$$N(t) = \begin{cases} N(t-1) + 1 & \text{if flip } t \text{ is a head,} \\ N(t-1) - 1 & \text{if flip } t \text{ is a tail.} \end{cases}$$

The simulated game then ends at the first value of t where $N(t) = \pm 3$, where the resulting sampling *observation* for the simulated experiment is $8 - t$, the amount won (positive or negative) for that play of the game.

The next example will illustrate these building blocks of a simulation model for a prominent stochastic system from queueing theory.

Example 2: An *M/M/*1 Queueing System

Consider the *M/M/*1 queueing theory model (Poisson input, exponential service times, and single server) that was discussed at the beginning of Sec. 15.6. Although this model already has been solved analytically, it will be instructive to consider how to study it by using simulation. To be specific, suppose that the values of the *arrival rate* λ and *service rate* μ are

$$\lambda = 3 \text{ per hour}, \qquad \mu = 5 \text{ per hour}.$$

To summarize the physical operation of the system, arriving customers enter the queue, eventually are served, and then leave. Thus it is necessary for the simulation model to describe and synchronize the arrival of customers and the serving of customers.

Starting at time 0, the simulation clock records the amount of (simulated) time t that has transpired so far during the simulation run. The information about the queueing system that defines its current status, i.e., the state of the system, is

$$N(t) = \text{number of customers in system at time } t.$$

The events that change the state of the system are the *arrival* of a customer or a service completion for the customer currently in service (if any). We shall describe the event generation mechanism a little later. The state transition mechanism is to

$$\text{Reset} \qquad N(t) = \begin{cases} N(t) + 1 & \text{if arrival occurs at time } t, \\ N(t) - 1 & \text{if service completion occurs at time } t. \end{cases}$$

There are two basic methods used for advancing the simulation clock and recording the operation of the system. We did not distinguish between these methods for Example 1 because they actually coincide for that simple situation. However, we now describe and illustrate these two **time advance mechanisms** (fixed-time incrementing and next-event incrementing) in turn.

With the **fixed-time incrementing** time advance mechanism, the following two-step procedure is used repeatedly.

Summary of Fixed-Time Incrementing

1. *Advance time* by a small *fixed amount.*
2. *Update the system* by determining what events occurred during the elapsed time interval and what the resulting state of the system is. Also record desired information about the performance of the system.

For the queueing theory model under consideration, only two types of events can occur during each of these elapsed time intervals, namely, one or more *arrivals* and one or more *service completions*. Furthermore, the probability of two or more arrivals or of two or more service completions during an interval is negligible for this model if the interval is relatively short. Thus the only two possible events during such an interval that need to be investigated are the arrival of one customer and the service completion for one customer. Each of these events has a known probability.

To illustrate, let us use 0.1 hour (6 minutes) as the small fixed amount by which the clock is advanced each time. (Normally, a considerably smaller time interval would be used to render negligible the probability of multiple arrivals or multiple service completions, but this choice will create more action for illustrative purposes.) Because both interarrival times and service times have an exponential distribution, the probability P_A that a time interval of 0.1 hour will include an *arrival* is

$$P_A = 1 - e^{-3/10} = 0.259,$$

and the probability P_D that it will include a *departure* (service completion), given that a customer was being served at the beginning of the interval, is

$$P_D = 1 - e^{-5/10} = 0.393.$$

To randomly generate either kind of event according to these probabilities, the approach is similar to that in Example 1. The computer again is used to generate a random number, but this time with multiple digits rather than one. Placing a decimal point in front of the number then makes it a *uniform random number* on (0, 1), that is, a random observation from the *uniform distribution* between 0 and 1. If we denote this uniform random number by r_A,

$$r_A < 0.259 \Rightarrow \text{arrival occurred},$$

$$r_A \geq 0.259 \Rightarrow \text{arrival did not occur}.$$

Similarly, with *another* uniform random number r_D,

$$r_D < 0.393 \Rightarrow \text{departure occurred},$$

$$r_D \geq 0.393 \Rightarrow \text{departure did not occur},$$

given that a customer was being served at the beginning of the time interval. With no customer in service then (i.e., no customers in the system), it is assumed that no departure can occur during the interval even if an arrival does occur.

Table 21.1 shows the result of using this approach for 10 iterations of the *fixed-time incrementing* procedure, starting with no customers in the system and using time units of minutes.

Table 21.1 **Fixed-Time Incrementing Applied to Example 2**

t, time (min)	$N(t)$	r_A	Arrival in Interval?	r_D	Departure in Interval?
0	0				
6	1	0.096	Yes	—	
12	1	0.569	No	0.665	No
18	1	0.764	No	0.842	No
24	0	0.492	No	0.224	Yes
30	0	0.950	No	—	
36	0	0.610	No	—	
42	1	0.145	Yes	—	
48	1	0.484	No	0.552	No
54	1	0.350	No	0.590	No
60	0	0.430	No	0.041	Yes

Step 2 of the procedure (updating the system) includes recording the desired measures of performance about the aggregate behavior of the system during this time interval. For example, it could record the *number of customers* in the queueing system and the *waiting time* of any customer who just completed his or her wait. If it is sufficient to estimate only the mean rather than the probability distribution of each of these random variables, the computer will merely add the value (if any) at the end of the current time interval to a cumulative sum. The sample averages will be obtained after the simulation run is completed by dividing these sums by the sample sizes involved, namely, the total number of time intervals and the total number of customers, respectively.

Next-event incrementing differs from fixed-time incrementing in that the simulation clock is incremented by a *variable* amount rather than by a fixed amount each time. This variable amount is the time from the event that has just occurred until the *next event* of any kind occurs; i.e., the clock jumps from event to event. A summary follows.

Summary of Next-Event Incrementing

1. *Advance time* to the time of the *next event* of any kind.
2. *Update the system* by determining its new state that results from this event and by randomly generating the time until the next occurrence of any event type that can occur from this state (if not previously generated). Also record desired information about the performance of the system.

For this example the computer needs to keep track of two future events, namely, the next arrival and the next service completion (if a customer currently is being served). These times are obtained by taking a random observation from the probability distribution of interarrival and service times, respectively. As before, the computer takes such a random observation by generating and using a random number. (This technique will be discussed in Sec. 21.2.) Thus, each time an arrival or service completion occurs, the computer determines how long it will be until the next time this event will occur, adds this time to the current clock time, and then stores this sum in a computer file. (If the service completion leaves no customers in the system, then the generation of the time until the next service completion is postponed until the next arrival occurs.) To determine which event will occur next, the computer finds the minimum of the clock times stored in the file. To expedite the bookkeeping involved,

simulation programming languages provide a "timing routine" that determines the occurrence time and type of the next event, advances time, and transfers control to the appropriate subprogram for the event type.

Table 21.2 shows the result of applying this approach through five iterations of the next-event incrementing procedure, starting with no customers in the system and using time units of minutes. For later reference, we include the *uniform random numbers* r_A and r_D used to generate the interarrival times and service times, respectively, by the method to be described in Sec. 21.2. These r_A and r_D are the same as those used in Table 21.1 in order to provide a truer comparison between the two time advance mechanisms.

The next-event incrementing procedure is considerably better suited for this example and similar stochastic systems than the fixed-time incrementing procedure. Next-event incrementing requires fewer iterations to cover the same amount of simulated time, and it generates a precise schedule for the evolution of the system rather than a rough approximation.

The next-event incrementing procedure will be illustrated again in Sec. 21.4 (see Table 21.12) in the context of a full statistical experiment for estimating certain measures of performance for another queueing system.

Several pertinent questions about how to conduct a simulation study of this type still remain to be answered. These answers are presented in a broader context in subsequent sections.

More Examples in Your OR Courseware

Simulation examples are easier to understand when they can be *observed in action*, rather than just talked about on a printed page. Therefore, the simulation area of your OR Courseware includes two *demonstration examples* under the Demo menu that should be viewed at this time.

Both examples involve a bank that plans to open up a new branch office. The questions address how many teller windows to provide and then how many tellers to have on duty at the outset. Therefore, the system being studied is a *queueing system*. However, in contrast to the *M/M/*1 queueing system considered in Example 2 above, this queueing system is too complicated to be solved analytically. This system has multiple servers (tellers), and the probability distributions of interarrival times and service times do not fit the standard models of queueing theory. Furthermore, in the second demonstration, it has been decided that one class of customers (merchants) needs to be given nonpreemptive priority over other customers, but the probability

Table 21.2 **Next-Event Incrementing Applied to Example 2**

t, time (min)	$N(t)$	r_A	Next Interarrival Time	r_D	Next Service Time	Next Arrival	Next Departure	Next Event
0	0	0.096	2.019	—	—	2.019	—	Arrival
2.019	1	0.569	16.833	0.665	13.123	18.852	15.142	Departure
15.142	0	—	—	—	—	18.852	—	Arrival
18.852	1	0.764	28.878	0.842	22.142	47.730	40.994	Departure
40.994	0	—	—	—	—	47.730	—	Arrival
47.730	1							