7

Simulation in the Urban Context

In urban operations research we often encounter the following situation: a new operational strategy has been proposed (e.g., a new strategy for allocating fire engines among firehouses or a different set of rules for ambulance dispatching) and we wish to investigate its implications for several indices of an urban service system's performance. In attempting to use for this purpose some of the analytical techniques that we have presented so far, either of two things may happen: the problem may prove to be too complicated to solve analytically; or the problem may be tractable in some respects but the level of detail provided by the analytical answers may be insufficient for our needs.

In either event, it is clear that we would like to learn much more about our problem. One possible solution is to put the new concept into practice on an experimental basis and see if it produces the desired results. However, many constraints of an economic, technical, legal, or political nature very often make such experiments infeasible. Only very rarely does one have an opportunity to perform large-scale experiments in the "real world" with an urban service system. The risks and the costs are usually so high that responsible administrators refuse to give permission or provide the resources necessary for such "trial runs."

Faced with an inability to use an analytical approach or to perform a real-world, real-time experiment with the actual system, the analyst or the planner might then resort to the technique of simulation.

Simulation is a term that has been used to describe many different kinds of activities. For instance, the military exercises that armed forces all over the world perform on a more-or-less regular basis can be considered as examples of simulation (of actual war conditions in this case). So, for that

matter, can that part of the training program of airline pilots which involves flying the mock version of an airplane (the "simulator") in a laboratory under "simulated" traffic conditions. Here, however, we shall use a considerably stricter definition:

> *Definition:* *Simulation* is the procedure in which a computer-based mathematical model of a physical system is used to perform experiments with that system by generating external stimuli ("demands") and observing how the system reacts to them over a period of time.

Thus, central to the idea of simulation—in this interpretation, at least—is the notion of a mathematical model. Almost equally important is the presence of the digital computer as the device which, through its prodigious computational abilities and speed, provides attractive and powerful features to simulation. In fact, it is because simulation is so closely linked with the computer as its tool that this technique did not really come on its own until after the second generation of computers came into existence in the early 1960s. Also central to the concept of simulation is the idea of learning from "empirical" evidence (i.e., from observation of the outcomes of experiments). This is in contrast to the classical approach in mathematics, in which problems are solved once and for all for some given set of assumptions. For this reason simulation is sometimes referred to as *the experimental branch of mathematics.*

The use of simulation in urban operations research is already considerable and is likely to continue growing in the future. In this chapter we shall examine those specific aspects of simulation which are particularly important in this area of application.

One such aspect is the simulation of probabilistic events. The importance of analyzing urban service systems from a probabilistic point of view is one of the central themes of this book: as we have already noted many times, it is the probabilistic nature of demand and of service requirements for most urban services that makes them difficult to manage and ultimately contributes heavily to their high cost. Thus, the ability to simulate random events is essential to simulation in urban operations research. Section 7.1 will deal in some detail with this topic.

The second area of emphasis in this chapter concerns techniques for working with two-dimensional geometrical relationships in the computer. Obviously, the ability to work with such relationships is essential to simulations of urban services. As we shall see, solutions to problems that appear "natural" and straightforward to human beings require careful analysis and delineation when programmed for a computer.

We shall also discuss briefly simulation languages and the technique of *event-paced simulation*, which is very convenient for simulating queueing systems of any kind. Finally, simulation is, in some instances, a rather controversial technique. We shall therefore conclude this chapter with a discussion of the advantages, disadvantages, and misuse of simulation as an urban operations research technique.

7.1 SIMULATING PROBABILISTIC EVENTS

Techniques for simulating "randomness" constitute an important part of the array of tools needed to carry out successfully a probabilistic simulation. These techniques have attracted considerable attention and have been developed to the point where it is possible to generate samples from virtually any probability distribution (or from reasonable approximations of these probability distributions).

The approach used to generate these samples consists of two steps. Sequences of statistically independent random numbers, *distributed uniformly* within some finite range, are first produced. These sequences are then processed and transformed into sequences of samples from the desired probability distributions. This second step is obviously the more interesting of the two: it is indeed surprising how a sequence of independent, uniformly distributed random numbers can be used to simulate almost any kind of random phenomenon imaginable.

7.1.1 Generating Random Numbers

In generating sequences of random numbers we wish to duplicate, in effect, a game in which a fair wheel of fortune is spun and the outcome of the game is recorded after each spinning. If the periphery of the wheel of fortune is divided into m equal intervals which are indexed with the integers from 1 to m, this game produces independent random integers which are uniformly distributed in the 1 to m range.

There are basically two ways of obtaining sequences of random numbers for a simulation experiment. The first way is simply to "read" the random numbers from a list of such numbers which has already been compiled by somebody else. For example, a table with 1 million random digits was published in 1955 by the Rand Corporation. Such tables of random numbers have been recorded on magnetic tape and can thus be read quickly by a digital computer.

The second way (and, by now, the most common) is to have the computer itself generate a sequence of random numbers. This is accomplished by having the computer execute a short "program" every time a new random number is needed. This computer program essentially uses the last random number produced, say the $(n-1)$st in the sequence, to produce the next random

number, say the nth in the sequence. The specific method employed can be any one of the *congruential methods*.

For instance, the *mixed* congruential method uses the expression

$$x_n = ax_{n-1} + c \qquad \text{(modulo } m\text{)} \tag{7.1}$$

where x_n and x_{n-1} are the nth and $(n-1)$st random numbers in the sequence, respectively, and a, c, and m are suitably chosen positive integers with $a < m$ and $c < m$. The indication "modulo m" means that x_n is the *remainder* of the division of the quantity $ax_{n-1} + c$ by the number m. For example, if $a = 5$, $x_{n-1} = 7$, $c = 3$, and $m = 16$, we have $ax_{n-1} + c = 38$ and $x_n = 38$ (modulo 16) $= 6$.

How many different numbers can we obtain through (7.1)? Obviously, *at most*, m, that is all the numbers between 0 and $m - 1$. For example, with $a = 5$, $c = 3$, $x_0 = 7$, and $m = 16$, we obtain $x_1 = 6$ (as we just saw) and then $x_2 = 1$, $x_3 = 8$, $x_4 = 11$, $x_5 = 10$, $x_6 = 5$, $x_7 = 12$, $x_8 = 15$, $x_9 = 14$, $x_{10} = 9$, $x_{11} = 0$, $x_{12} = 3$, $x_{13} = 2$, $x_{14} = 13$, and $x_{15} = 4$, in all the 16 different numbers in the range from 0 to 15. What is x_{16} in this sequence? We have $a \cdot x_{15} + c = 23$ and $x_{16} = 23$ (modulo m) $= 7$. Thus, the earlier sequence will now be repeated once again with $x_{17} = 6$, $x_{18} = 1$, and so on.

The example above illustrates two interesting points. First, any sequence of random numbers produced through (7.1) is cyclical. Each cycle consists of a necessarily finite number of distinct numbers (at most m). After a cycle has been completed, a new cycle identical to the previous one begins. (In our example each cycle consists of 16 distinct numbers.) The finiteness of the cycles will obviously cause problems in a simulation if the length of a cycle is small: a succession of identical short cycles of numbers definitely does not behave as a sequence of independent random numbers (we have $x_{n+\tau} = x_n$, where τ is the length of the cycle). However, if m is a very large number and a and c are chosen with sufficient care to make the length of each cycle comparable to the size of m, the finiteness of the cycles is of no practical significance. Consider the case when m is chosen to be equal to 2^b, where b is the number of bits in a binary computer word. (This is the choice of m made in computer-based simulations, since 2^b is the total number of integers that can be expressed in binary form with the number of bits available.) When $b = 32$, we have $m = 2^{32} \approx 4.3$ billion distinct numbers. With a cycle length in this order of magnitude, it is an entirely academic matter that the same sequence of numbers will be repeated at some future point.

The second observation concerns the meaning of the word "random" in the case of sequences produced through (7.1). Obviously, if we know a, c, and m we can predict perfectly the complete sequence that will follow any initial number x_0. For this reason, the initial number x_0 used to produce some sequence of numbers is called the *seed* of this sequence. For the same

reason, the sequences of numbers produced through (7.1) (as well as through other congruential methods) are also called *pseudo-random* numbers. This, however, is also of academic importance as long as the sequences of numbers produced through (7.1) qualify (through passing the appropriate statistical tests) as independent samples from a discrete, uniform probability distribution, that is, as long as the successive numbers appear to an observer to be drawn from a game similar to the spinning wheel game that we described earlier.

In fact, the property of reproducibility for the sequences generated through (7.1) is in itself a most desirable one. For, when we wish to perform a simulation experiment under "identical conditions" with some earlier experiment, all we have to do is provide the same seed, x_0, used in the earlier experiment to obtain the same sequence of random (or pseudo-random) phenomena as before.

We have yet to say how the constant positive integers a and c in (7.1) are chosen. An unfortunate choice of a and c will unavoidably lead to short cycles[1] of numbers (which are usually anything but uniformly distributed) even for a large m. Happily, the branch of mathematics called number theory provides the guidelines for a good choice of a and c. For instance, in the case of digital computers, where, as we have mentioned, we use $m = 2^b$, it can be shown that a should be chosen such that it is equal to 1 in modulo 4 arithmetic (i.e., $a = 1, 5, 9, 13, \ldots$) and that c should be an odd number.[2] The choice of x_0, the seed, is immaterial in this case as far as the length of the cycle is concerned.

In summary, the desirable properties of any method for producing sequences of random numbers in a digital computer are:

1. The numbers should appear to be statistically independent of each other (although, strictly speaking, they are perfectly correlated).
2. The numbers should be uniformly distributed over some range.
3. The sequence of numbers should not be self-repeating for any desired length.
4. The random numbers should be producible at very high speed.
5. The method should place minimal requirements on the memory of the computer.
6. Any sequence of random numbers obtained during a given simulation experiment should be reproducible.

[1]Try, for instance, $a = c = 8$ and $m = 16$ in (7.1). Observe how, for any choice of x_0, (7.1) will very soon begin producing just the number 8.

[2]Strictly speaking, these are necessary but not sufficient conditions for the sequences of random numbers to possess all the statistical properties of independent samples from a uniform distribution.

The congruential method that we have just described is typical of the techniques used to generate random numbers and possesses all of the properties listed above.

In a practical sense, the important fact is that any modern computer system is preprogrammed to produce sequences of random numbers that possess—at least as a close approximation—all six of the desirable properties mentioned above. Typically, these numbers are produced by just calling an appropriately named "function" or "subroutine" in the computer repertoire (typical names for these miniprograms include RAND, RANDU, etc.). All that is required of the programmer is to provide a seed to begin the sequence. The computer subsequently provides automatically the input (i.e., the number x_{n-1} in the mixed congruential method) needed to produce the next number, x_n, in the sequence.

The random numbers produced through these preprogrammed methods are usually presented in the form of numbers uniformly distributed between 0.0 and 1.0 [which is done by dividing internally the random number resulting from (7.1) by the number m]. Obviously, the 0 to 1 interval is thus subdivided so finely that, for all practical purposes, it can be assumed that the computer produces statistically independent samples from the continuous uniform probability density function shown on Figure 7.1. This, too, will be our assumption from here on, and we shall use the expression "independent uniformly distributed (i.u.d.) over [0, 1]" random numbers.

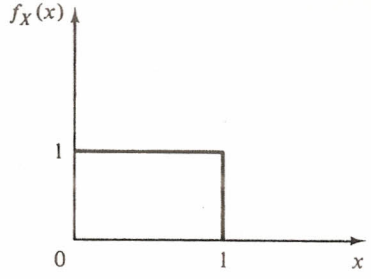

FIGURE 7.1 The random variable X with the uniform pdf on [0, 1] is an idealized model of the random number R, samples of which are produced by random number generators.

7.1.2 Using Several Random-Number Generators

It is often advisable to design a simulation program so that it uses different independent random-number generators to produce different sequences of random numbers. This is easy to accomplish with most modern computer systems and enhances greatly the "controllability" of the experiments that one may wish to perform with a simulation program.

Consider, for example, the task of simulating demands for a spatially distributed urban service system. Assume that the times when such demands occur are statistically independent from the locations of these demands and that demands are served in a FCFS manner. Given then pdf's for the time distribution and for the spatial distribution of demands, it is advisable to use one random-number generator, A, to simulate the instants when demands occur and a different random-number generator, B, to simulate demand locations. For it may be desirable, in this case, to perform experiments in which, say, the *times* when demands occur change from one experiment to another while the sequence of the locations of demand remains the same. One way such experiments can be performed with two random-number generators is by using one single seed for B for *all* experiments while using a different seed for A for each experiment. Indeed, it is not unusual to find urban system simulations that use 10 or more independent random-number generators.

7.1.3 Generating Samples from Probability Distributions

We now turn to a discussion of how to generate sample values (i.e., random observations) of specific random variables. There are at least four different ways of doing this. Throughout this section it will be assumed that we have access to a source of "i.u.d. over [0, 1]" random numbers. We shall denote these numbers by r (or r_i, when it is necessary to use an index). Thus, r is a sample value of the random variable R with pdf

$$f_R(r) = \begin{cases} 1, & \text{if } 0 \leq r \leq 1 \\ 0, & \text{otherwise} \end{cases}$$

Inversion method. We first consider the most fundamental of the techniques for generating sample values of random variables. It can be applied, at least in principle, in all cases where an explicit expression exists for the cumulative distribution function of the random variable. Basically, the method depends on the fact that, for any random variable Y, the cdf, $F_Y(y)$, is a nondecreasing function with $0 \leq F_Y(y) \leq 1$ for all values of y. This suggests a "match" of the cdf with the random numbers, r_i. Specifically, by drawing a random number r (see Figure 7.2) and then by finding the inverse image of r on the abscissa through

$$y_s = F_Y^{-1}(r) \tag{7.2}$$

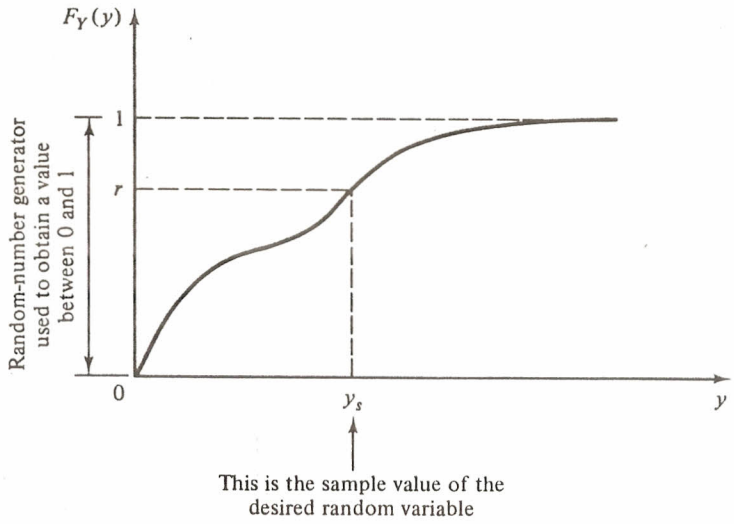

FIGURE 7.2 Illustration of the inversion method.

a sample value, y_s, of random variable Y can be generated. To understand why (see also Figure 7.2) observe that

$$P\{Y \leq y_s\} = P\{R \leq F_Y(y_s)\} = F_Y(y_s) \tag{7.3}$$

the last equality following from the fact that r is uniformly distributed in [0, 1]. The following examples illustrate the inversion method.

Example 1: Negative Exponential Random Variable

We have seen that the negative exponential random variable is by far the most common model for the time between urban incidents requiring service. It is therefore essential that we be able to generate random sample values, t_s, of the random variable X with the pdf

$$f_X(t) = \lambda e^{-\lambda t} \qquad \text{for } t \geq 0$$

As we know, the cumulative distribution of X is

$$F_X(t) = \int_0^t \lambda e^{-\lambda \tau}\, d\tau = 1 - e^{-\lambda t} \qquad \text{for } t \geq 0$$

Let us then set a random number r_1 (uniformly distributed between 0 and 1) equal to $F_X(t)$. We have

$$r_1 = F_X(t) = 1 - e^{-\lambda t}$$

or, equivalently,

$$t = \frac{-\ell n\,(1 - r_1)}{\lambda} \tag{7.4}$$

Note that (7.4) allows us to draw sample observations of X. That is, each time we wish to draw a sample t_s, all we have to do is draw a random number r_1 and use (7.4). In fact, since $1 - r_1$ is also a random number uniformly distributed between 0 and 1, we might as well bypass the subtraction and use directly the random number generated by the computer. Thus, we finally have

$$t_s = \frac{-\ell n\,(1 - r_1)}{\lambda} = \frac{-\ell n\, r_2}{\lambda} \tag{7.5}$$

where we have used a second random number r_2 as a substitute for $1 - r_1$.

The procedure that we have used is illustrated in Figure 7.3. All we do is draw a random number between 0 and 1 and then find its "inverse image" on the t-axis by using the cdf. For a numerical example, let us suppose that we have $\lambda = 2$ and that we draw the random number $r = 0.168$. Then

$$t_s = \frac{-\ell n\,(0.168)}{2} \approx 0.892$$

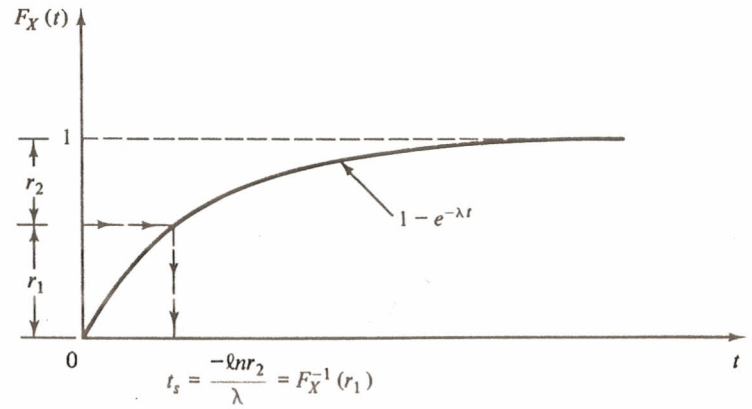

FIGURE 7.3 Generation of sample values from a negative exponential pdf through the inversion method.

Example 2: Locations of Accidents on a Highway

Consider a 6-mile stretch of a highway, lying between the fourth- and tenth-mile points of the highway. The spatial incidence of accidents on this 6-mile stretch is shown in Figure 7.4, which indicates that the second half of that stretch is twice as "dangerous" as the first half. We wish to simulate the locations of accidents on the stretch; the locations will be denoted with the random variable X, which is measured from the beginning of the highway at $X = 0$.

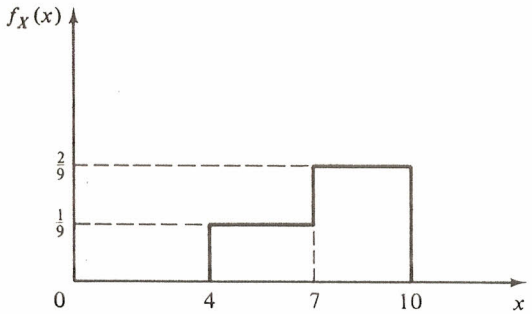

FIGURE 7.4 Probability density function for X.

For the cdf of X, we have

$$F_X(x) = \int_{-\infty}^{x} f_X(\xi)\, d\xi = \begin{cases} 0 & \text{for } x < 4 \\ \frac{1}{9}(x-4) & \text{for } 4 \le x < 7 \\ \frac{2}{9}(x-7) + \frac{1}{3} = \frac{1}{9}(2x-11) & \text{for } 7 \le x < 10 \\ 1 & \text{for } 10 \le x \end{cases}$$

This is plotted in Figure 7.5. In the interval of interest, [4, 10], $F_X(x)$ has a breakpoint at $x = 7$. At that point $F_X(x) = \frac{1}{3}$. Thus, for values of a random number, r, which are less than 1/3, we should use $r = F_X(x_s) = \frac{1}{9}(x_s - 4)$ or $x_s = F_X^{-1}(r) = 9r + 4$, while otherwise we should use $x_s = F_X^{-1}(R) = \frac{1}{2}(9r + 11)$. This procedure is illustrated in Figure 7.6. Note that the procedure uses a single random number to generate each sample observation, but requires some algebraic work.

We can also use an approach that requires no such work if we are willing to generate two random numbers rather than one for each sample observation. This alternative approach is shown by the flowchart of Figure 7.7. Therefore, if the generation of random numbers is not particularly time-consuming (and nowadays it is not), the alternative approach of Figure 7.7 would probably be chosen over the one of Figure 7.6. Many variations, in terms of particular details, of either approach may also be visualized—all leading to a satisfactory generation of random observations of random variable X.

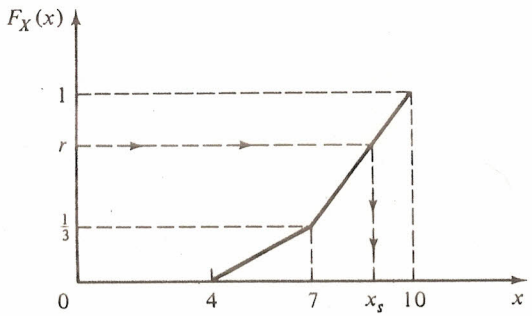

FIGURE 7.5 Cumulative distribution for X.

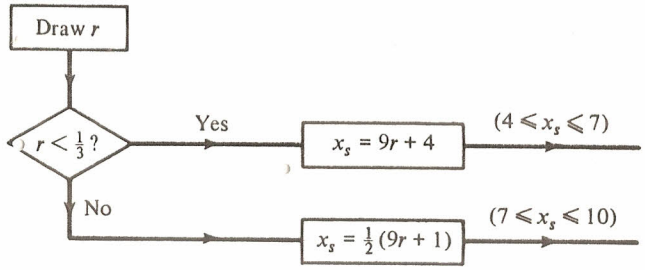

FIGURE 7.6 Generation of random observations of random variable X.

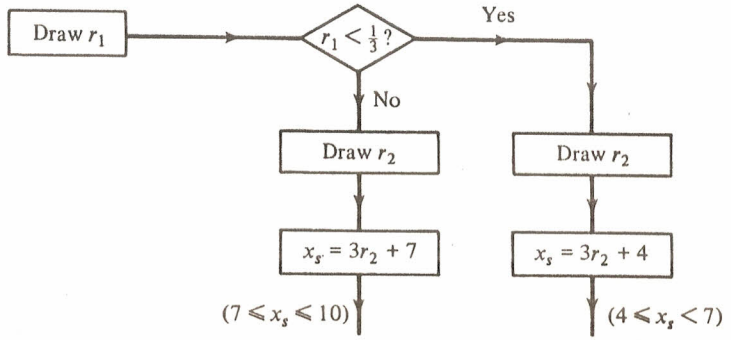

FIGURE 7.7 Alternative way of generating random observations of random variable X.

In an identical manner, we can use the inversion method to generate samples from discrete random variables. This is shown in Figure 7.8. If $\{x_1, x_2, x_3, \ldots\}$ are the values that the discrete random variable X can take and $p_X(x_k)$ and $P_X(x_k)$ are the pmf and cdf, respectively, of X, the method consists of (1) drawing a random number r, and (2) finding the smallest value x_i such that

$$P_X(x_i) = \sum_{\text{all } x_k \le x_i} p_X(x_k) \ge r \tag{7.6}$$

In that case we set $x_s = x_i$ (where x_s denotes the sample value of X). Note that entire ranges of values of r now correspond to a single value of X.

For instance, if X has a *geometric* pmf,

$$p_X(k) = p(1-p)^{k-1} \qquad k = 1, 2, \ldots$$

then sample values of X can be obtained from

$$k_s = \left\lceil \frac{\ell n\,(1-r)}{\ell n\,(1-p)} \right\rceil \tag{7.7}$$

where $\lceil a \rceil$ denotes "the smallest integer that is greater than or equal to a."

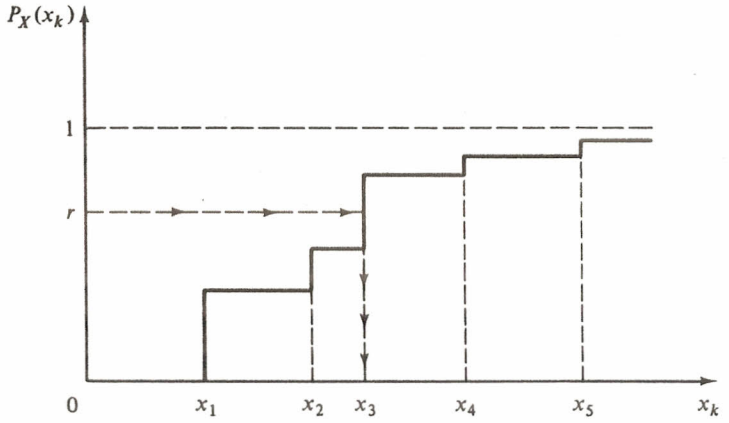

FIGURE 7.8 Use of the inversion method for sampling from discrete distributions.

Exercise 7.1 Derive (7.7).

In general, the inversion method is useful in generating sample values of random variables whose cdf's are relatively easy to work with. Many other random variables, whose cdf's do not fall into this category, can be expressed as functions of these "simple" random variables. Therefore, the inversion method is also important as a "building block" in more sophisticated approaches, such as those described next.

Relationships method. A second approach for generating sample observations of a given random variable is based on taking advantage of some known relationship between this random variable and one or more other random variables. This method has been widely exploited in the case of some of the best-known (and most often used in urban operations research) pmf's and pdf's, such as the Erlang, the binomial, and the Poisson.

Example 3: kth-Order Erlang PDF

We know that the kth-order Erlang random variable X, with pdf

$$f_X(x) = \frac{\lambda^k x^{k-1} e^{-\lambda x}}{(k-1)!} \qquad \text{for } x \geq 0,\ k = 1, 2, 3, \ldots$$

can be considered to be the sum of k independent, identically distributed random variables, each with negative exponential pdf with parameter λ. That is,

$$X = T_1 + T_2 + \ldots + T_k$$

with $f_{T_i}(t) = \lambda e^{-\lambda t}$ for $t \geq 0$.

Thus, a simple way to generate random observations of X is to generate k random observations from a negative exponential pdf by using (7.5) and then to add these k observations to obtain a *single* observation of X. That is,

$$x_s = \sum_{i=1}^{k} t_{si} = \sum_{i=1}^{k} \left(-\frac{\ell n\, r_i}{\lambda}\right) = -\frac{1}{\lambda} \sum_{i=1}^{k} \ell n\, r_i = -\frac{1}{\lambda}\, \ell n \left\{\prod_{i=1}^{k} r_i\right\} \qquad (7.8)$$

The rightmost of these expressions is the most efficient one for computational purposes.[3]

Example 4: Binomial PMF

The binomial probability mass function indicates the probability of K successes in n Bernoulli trials. To obtain sample values of K, we simply simulate n Bernoulli trials in the computer. That is, we obtain n random observations of a discrete random variable with probability of "success" and of "failure" equal to p and to $1 - p$, respectively, and then count the total number of successes in the n trials. This total number of successes is the sample value of K.

Example 5: Simulating Poisson Events

Consider the Poisson pmf $P\{N(T) = k\}$, which can be thought of as indicating the probability of observing k events in a time interval T when interarrival times are independent with negative exponential distribution. To generate random observations of $N(T)$ for any given T, we follow the procedure shown on Figure 7.9. We keep generating exponentially distributed time intervals $t_{s1}, t_{s2}, t_{s3}, \ldots$ [by using (7.5)] until the total length of time represented by the sum of these intervals exceeds T for the first time. That is, we find j such that

$$\sum_{i=1}^{j} t_{si} \leq T < \sum_{i=1}^{j+1} t_{si}$$

Then our sample observation of $N(T)$ is given by $k_s = j$.

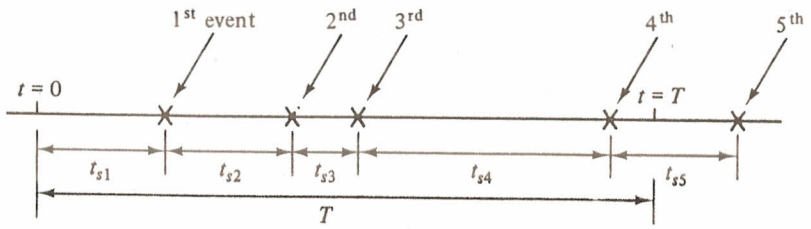

FIGURE 7.9 Generation of random observations from Poisson distribution. In this figure, $k_s = 4$.

[3] t_{si} in (7.8) denotes a sample observation of the negative exponential random variable T_i.

Note how in Examples 3–5 the inversion method (for generating sample values of negative exponential or of Bernoulli random variables) is used as a building block (or "subroutine") in the overall procedure.

The rejection method. The rejection method is a quite ingenious approach and is based on a geometical probability interpretation of pdf's (cf. Section 3.3.2). It can be used for generating sample values for any random variable that:

1. Assumes values only within a finite range.
2. Has a pdf that is bounded (i.e., does not go to infinity for any value of the random variable).

Let X be such a random variable. Let the maximum value of the pdf $f_X(x)$ be denoted as c and let X assume values in the range $[a, b]$ (X need not assume all possible values in $[a, b]$). To generate random observations of X through the rejection method (see Figure 7.10):

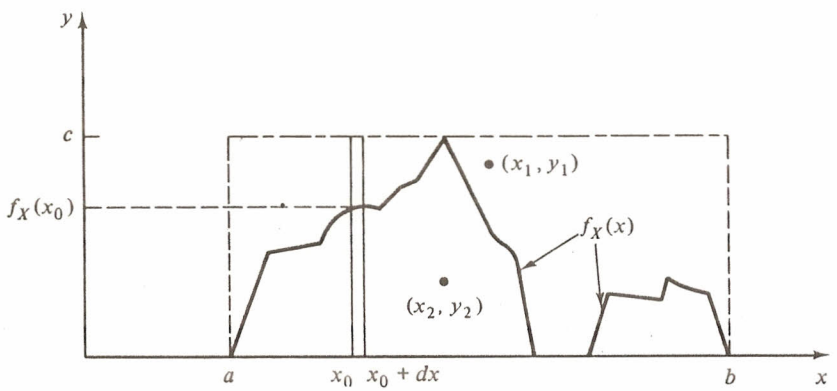

FIGURE 7.10 Using the rejection method for generating sample values from a random variable X.

STEP 1: Enclose the pdf $f_X(x)$ in the smallest rectangle that fully contains it and whose sides are parallel to the x and y axes. This is a $(b - a) \times c$ rectangle.[4]

STEP 2: Using two random numbers, r_1 and r_2, and scaling each to the appropriate dimension of the rectangle [by multiplying one by $(b - a)$ and the other by c] generate a point that is uniformly distributed over the rectangle.

[4]The method will actually work with any rectangle that contains this smallest rectangle. However the probability of rejection will then be greater than with the $(b - a) \times c$ rectangle.

STEP 3: If this point is "below" the pdf, *accept* the x-coordinate of the point as an appropriate sample value of X. Otherwise, *reject* and return to Step 2.

The validity of the foregoing procedure is not intuitively obvious on a first reading. We illustrate it first through an example and then explain why it works.

Example 6: Locations of Accidents on a Highway, Revisited

Consider once more the problem of simulating the location of accidents on a 6-mile stretch of highway (Example 2). For the random variable X with pdf $f_X(x)$ (Figure 7.4), we have $c = \frac{2}{9}$, $a = 4$, and $b = 10$.

Suppose now that the pair of random numbers $r_1 = 0.34$, $r_2 = 0.81$ is drawn in Step 2. These random numbers, with appropriate scaling, identify the point $(x_1, y_1) = [(10 - 4)(0.34) + 4, \frac{2}{9}(0.81)] = (6.04, 0.18)$ on the x-y plane. Since at $x_1 = 6.04$, $f_X(x_1) = \frac{1}{9} \doteq 0.11$ (see Figure 7.4), the point (x_1, y_1) is found in Step 3 to be above $f_X(x)$ and is rejected. No sample value of X has been obtained in this trial.

Returning to Step 2, suppose that the new pair of random numbers turns out to be $r_1 = 0.41$, $r_2 = 0.15$. This results in the point $(x_2, y_2) \doteq (6.46, 0.033)$, which is below $f_X(x)$. Therefore, the sample value $x_s = 6.46$ is accepted.

It is easy to see in this case that the only pairs of random numbers that will be rejected are those for which $r_1 < 0.5 \leq r_2$.

The reason why this method works is quite simple. The points (x, y) obtained through the procedure of Step 2 are uniformly distributed over the area of the rectangle, $(b - a) \times c$. Therefore, for any point whose x-coordinate is between x_0 and $x_0 + dx$ (see Figure 7.10), we have

$$P\{\text{point is accepted} \mid x_o \leq x \leq x_o + dx\} = \frac{f_X(x_0)\,dx}{c\,dx} = \frac{f_X(x_0)}{c} \qquad (7.9)$$

Since the x-coordinate of the points (x, y) is chosen randomly in $[a, b]$, the sample observations that will be accepted will then be distributed exactly as $f_X(x)$. In other words, were we to obtain a very large number of sample observations, x_s, and then construct a histogram of these observations, the histogram would tend to duplicate the shape of $f_X(x)$.

Note that the rejection method is a very general one since it makes only the two assumptions that we have mentioned regarding the functional form of $f_X(x)$. It is especially convenient and easy to use with pdf's which are shaped like polygons. Its main disadvantage is that for some pdf's it may take many rejected pairs of random numbers before an accepted sample value is found.

Exercise 7.2 Show that

$$E[\text{number of trials until an accepted sample value is found}] = c(b - a) \qquad (7.10)$$

Note: This is independent of the functional form of $f_X(x)$!

Method of approximations. The fourth approach to generating sample values of random variables consists simply of approximating in some way a probability distribution by another distribution which is simpler to simulate through one of the three earlier methods.

The classical example of this type of approach is the generation of random observations from a Gaussian (normal) distribution.

Example 7

Generate random observations of a random variable X with a Gaussian distribution, mean μ_x and standard deviation σ_x.

Solution

No closed-form expression exists for the cumulative distribution $F_X(x)$ of a Gaussian random variable. Consequently, short of an approximate numerical procedure, we cannot use the inversion method and must resort to some other technique. One approach that can be made arbitrarily accurate and is easy to implement is based on the Central Limit Theorem (CLT):

Define a random variable Z as the sum of k independent random variables uniformly distributed between 0 and 1:

$$Z = R_1 + R_2 + R_3 \ldots + R_{k-1} + R_k$$

Now from the CLT we know that for large k, Z has approximately a Gaussian distribution. Since for each R_i we have $E[R_i] = 1/2$ and $\sigma_{R_i} = 1/\sqrt{12}$, the mean value of Z is $k/2$ and its standard deviation $\sqrt{k/12}$. Thus, we can use

$$\frac{X - \mu_x}{\sigma_x} \approx \frac{Z - k/2}{\sqrt{k/12}} \qquad \text{for large } k \qquad (7.11)$$

since both sides of this equation are equal to a "normalized" Gaussian random variable with mean 0 and standard deviation equal to 1. From (7.11) we finally have

$$x_s \cong \sigma_x\left(\frac{z - k/2}{\sqrt{k/12}}\right) + \mu_x = \frac{\sigma_x}{\sqrt{k/12}}\left(\sum_{i=1}^{k} r_i - \frac{k}{2}\right) + \mu_x \qquad (7.12)$$

where x_s is a random observation of random variable X. Generally, values of k greater than 6 or so provide adequate approximations to the Gaussian pdf

for values of X within two standard deviations of the mean. The most commonly used value is $k = 12$, since it simplifies somewhat the computation of (7.12). This means that we have to draw 12 random numbers $r_1, r_2, \ldots, r_{12}$ in order to obtain a single sample value x_s of X.

In general, the approximations approach is not always quite as sophisticated as our last example might indicate. Most often, in fact, the approach consists of simply approximating a complicated pdf (or cdf) with, say, a piecewise linear pdf (or cdf) and then using the inversion or the rejection method to generate random observations.

Final comment. Methods for generating samples from some of the standard pdf's and pmf's have been the subject of much research in recent years. Our main purpose above was to present some typical (and correct) approaches rather than the most accurate or efficient method in each example. For instance, excellent alternatives to (7.5) exist for simulating negative exponential pdf's [NEUM 51], [FISH 78].

Similarly, an alternative to (7.12) for the Gaussian random variable with mean μ_x and variance σ_x is the following:

1. Generate two random numbers r_1 and r_2.
2. Set:

$$\rho = \sigma_x\sqrt{-2\,\ell n\, r_1} \qquad (7.13)$$

$$\theta = 2\pi r_2 \qquad (7.14)$$

3. Obtain samples, x_s, of the Gaussian random variable by setting

$$x_s = \mu_x + \rho \sin\theta \qquad (7.15)$$

This method is *exact* and requires only two random numbers. It can be derived from the analysis described in Example 3 of Chapter 3 (that involved the Rayleigh distribution) and is further developed in Problem 7.2.

Many of the more advanced simulation languages/systems (e.g., GPSS, SIMSCRIPT) provide standard internal subroutines for the generation of samples from some of the most common pdf's and pmf's (e.g., the Gaussian, the negative exponential, the Poisson, and the uniform): all that a user of the language/system must do is provide the necessary input parameters. The subroutine then automatically returns a sample value from the specified random variable.

Review example. We review now the first three of the foregoing approaches with an example.

Example 8: Triangular PDF

A hospital is located at the *center* of a district 1- by 1-mile square. Travel is parallel to the sides of the square, and emergency patient demands are distributed uniformly over the square. We wish to generate sample values of the total distance between an emergency patient and the hospital. The pdf for the total distance $D = D_x + D_y$ is shown in Figure 7.11.

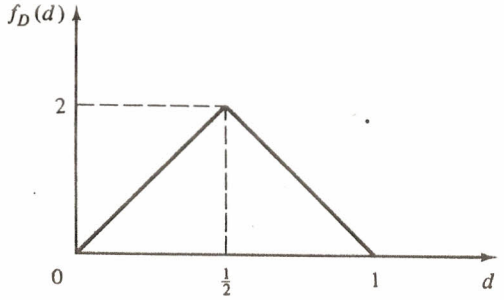

FIGURE 7.11 PDF for the distance D in Example 8.

Solution

1. To use the inversion method we first obtain the cdf,

$$F_X(x) = \begin{cases} 0 & \text{for } x < 0 \\ 2x^2 & \text{for } 0 \leq x < \frac{1}{2} \\ 4x - 2x^2 - 1 & \text{for } \frac{1}{2} \leq x < 1 \\ 1 & \text{for } 1 \leq x \end{cases}$$

Then the following "algorithm" suggests itself:

STEP 1: Draw a random number r. If $0 \leq r < \frac{1}{2}$, go to Step 2. If $\frac{1}{2} \leq r < 1$, go to Step 3.

STEP 2: From $r = F_X(x) = 2x^2$, we have $x_s = F_X^{-1}(r) = \sqrt{r/2}$. This generates samples, x_s, with values between 0 and 1/2.

STEP 3: From $r = F_X(x) = 4x - 2x^2 - 1$, we have $x_s = F_X^{-1}(r) = 1 - \sqrt{(1-r)/2}$. This generates samples, x_s, with values between 1/2 and 1.

Note that in computing the value of x_s in Steps 2 and 3, we have rejected the solutions $x_s = -\sqrt{r/2}$ (in Step 2) and $x_s = 1 + \sqrt{(1-r)/2}$ which would have provided sample observations outside [0, 1] (i.e., for nonlegitimate values of x_s).

2. If we decide to search for a relationship between random variable X and some other random variable(s), we should be able to note immediately that the pdf $f_X(x)$ is identical to the one that would have resulted from the definition

$$X \triangleq \frac{Y_1 + Y_2}{2}$$

where Y_1 and Y_2 are independent, identically distributed random variables with uniform distributions in [0, 1]. (Readers should convince themselves that this is so.) Therefore, a very simple method for generating random observations of X is to use

$$x_s = \frac{r_1 + r_2}{2}$$

where r_1 and r_2 are the usual random numbers obtained through the random-number generator. This is equivalent to generating samples of the distance traveled in the x-direction, D_x, and in the y-direction, D_y, and adding the two.

3. In using the rejection method, suppose that the first pair of random numbers drawn are $r_1 = 0.84$ and $r_2 = 0.27$. This results in the point $(x_1, y_1) = [1 \cdot (0.84), 2 \cdot (0.27)] = (0.84, 0.54)$ in the x-y plane. Since at $x_1 = 0.84$, $f_X(0.84) = 4(1 - 0.84) = 0.64$ (> 0.54), the sample observation is acceptable and we set $x_s = 0.84$. Note that in this case 50 percent of the pairs will be accepted on the average.

4. "Most natural": Generate a random point $(x = r_1, y = r_2)$ in the square and compute

$$D = |r_1 - \tfrac{1}{2}| + |r_2 - \tfrac{1}{2}|$$

7.1.4 Simulating Event Times from a Time-Dependent Poisson Process

Suppose that, as in Section 4.11, we have demands for an urban service which are generated in a Poisson manner but with an average rate, $\lambda(t)$, which is a function of time. This is a most common case for urban service systems. We would, therefore, like to be able to simulate this type of demand accurately and efficiently for any given time interval $[0, T]$. In the following it will be assumed that the average demand rate $\lambda(t)$ is known for all t $(0 \leq t \leq T)$. A typical pattern for $\lambda(t)$ is shown in Figure 7.12.

The most obvious approach is to use (7.5), substituting $\lambda(t)$ for λ. We begin at time $t = 0$ and generate the first demand using $\lambda(0)$ for λ in (7.5). Assume that the first demand occurs at time t_1. We then use (7.5) with $\lambda(t_1)$ to generate the time for the second demand, t_2. We proceed in the same way,

using $\lambda(t_2)$, $\lambda(t_3)$, . . . successively in (7.5) until eventually a demand's time of occurrence falls outside the interval T, at which point we can stop.

On second thought, however, this procedure turns out to be faulty. To see why, consider the case when a demand is simulated to occur at time t^*, as shown in Figure 7.12. To generate the next demand we would then use $\lambda(t) = \lambda^*$ with (7.5). But because λ^* is small, the simulated interval until the next demand will tend to be a long one (its expected value is $1/\lambda^*$), and thus we may "skip" (without generating any new demands) a good part of the period of time that follows t^*, when there is an increase (a "hump") in demand. In the extreme case when λ^* is close to 0, we may in fact "jump" all the way to the end of the interval T.

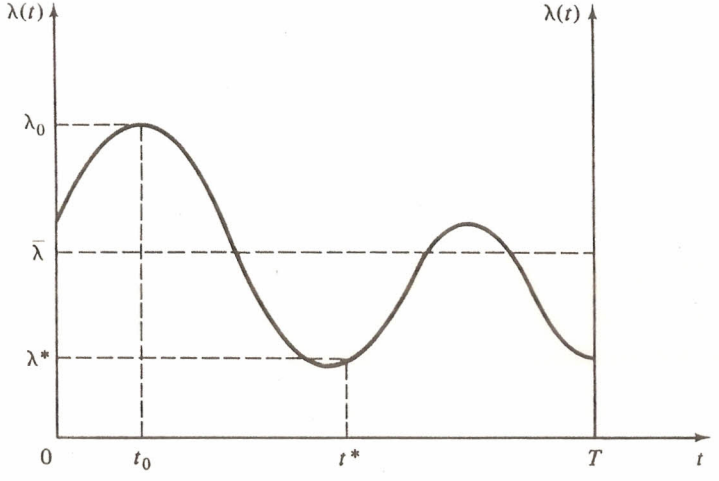

FIGURE 7.12 Time-varying arrival rate for a Poisson process.

The following four-step procedure overcomes these problems:

STEP 1: Estimate the mean arrival rate throughout the interval T,

$$\bar{\lambda} = \frac{1}{T}\int_0^T \lambda(t)\,dt \tag{7.16}$$

STEP 2: Using a *constant rate* $\bar{\lambda}$ with (7.5), generate a sample value of $N(T)$, say k, the number of Poisson events in T, in a manner identical to that of Example 5 (and Figure 7.9). Note that k is distributed as

$$P\{N(T) = k\} = \frac{(\bar{\lambda}T)^k e^{-\bar{\lambda}T}}{k!} \qquad k = 0, 1, 2, \ldots \tag{7.17}$$

STEP 3: Use the rejection method and the function $\lambda(t)$ with $c = \lambda_0$ (Figure 7.12), $a = 0$, and $b = T$ to generate *exactly k unordered* time instants $\{\tau_1, \tau_2, \tau_3, \ldots, \tau_k\}$ in $[0, T]$. These are the instants when demands occur (cf. "unordered arrival times" for a Poisson process, Section 2.12). Note that τ_i can be greater or less than τ_{i+1}.

STEP 4: Rearrange (sort) the time instants $\{\tau_1, \tau_2, \ldots, \tau_k\}$ into an ordered set $\{t_1, t_2, \ldots, t_k\}$ where t_1 is equal to Min $\{\tau_1, \tau_2, \ldots, \tau_k\}$, t_2 is equal to the second smallest of $\{\tau_1, \tau_2, \ldots, \tau_k\}$, and so on. The instants $\{t_1, t_2, \ldots, t_k\}$ are the simulated time instants for the occurrence of demands according to the time-dependent Poisson process with average rate $\lambda(t)$.

Exercise 7.3 Argue why the four-step procedure outlined above works correctly. What does each one of the four steps of the procedure accomplish?

The procedure presented above is somewhat inefficient because it makes two passes over T: once to generate k and a second time to generate the time instants, τ_i. A single-pass procedure is developed in Problem 7.3. We also note that an efficient sorting algorithm that can be useful for Step 4 is described in Section 7.2.1.

7.2 GEOMETRICAL RELATIONS IN URBAN SIMULATION

In the course of simulating urban service systems, questions of the following type arise repeatedly:

1. Given the coordinates of an incident requiring dispatching of a service unit, in which geographical or administrative subdivision did this incident occur?
2. Which of an agency's available service units is the closest one (in terms of travel distance or travel time) to a request for assistance?
3. Which ambulance zones have areas in common with a particular police precinct?
4. Which city blocks lie closest to each one of a new set of voting centers, so that voters can be assigned to the voting precinct most convenient to them?
5. Which ZIP-code areas in a city contain segments of a particular highway or railroad?

6. Given that a refuse incinerator is to be installed at a particular location and that it has a certain geometrically described pattern of smoke dispersal, which voting districts will be affected by the pollutants?

7. Which geographical subdivisions of a city have overlapping parts with the various noise contours (reflecting different noise levels) that will result from a new runway that the local airport authority is contemplating?

These questions and many others of a similar nature are obviously concerned with two-dimensional geometrical relations in urban environments. Most of them could be answered by inspection by a person equipped with a detailed map. However, the computer does not possess the visual and conceptual capabilities of human beings and so must be "told" in precise algorithmic terms how to answer these questions. It is important, moreover, that these algorithms be as efficient as possible, for it may be necessary for a computer to answer questions such as (1) or (2) above, several million times in the course of a typical set of simulation experiments.

In this section we shall discuss such techniques and algorithms. Most of them are of interest by themselves, independently of the simulation context, for they can be used with any computer package that processes geographically oriented data (e.g., "GEO-CODED" data). We shall begin our presentation with a seemingly unrelated problem, that of sorting a list of numbers and of searching through that list.

7.2.1 Efficient Sorting and Searching

In simulating urban systems it is often necessary to *sort* long arrays of numbers such as the coordinates of incidents, the addresses of mail recipients, or the times at which requests for service occurred. Equally important, sorting algorithms play a central role as "building blocks" in techniques aimed at resolving several important geometrical problems, as we shall see shortly. It is therefore important to develop a sorting algorithm that is as efficient as possible.

Suppose that we are given n integers and that we wish to sort them in, say, increasing order of magnitude with the smallest number at the top of the list. The straightforward approach is to scan the initial list of numbers in order to find the smallest of them; this requires $n - 1$ comparisons. We make this smallest number the first element of our sorted array. We then scan the list of the remaining $n - 1$ numbers for the second smallest number in the initial list (thus performing $n - 2$ comparisons) and continue in this manner until all the numbers are sorted. This is, then, a $O(n^2)$ algorithm, since the

total number of comparisons required is

$$\sum_{i=1}^{n} (i - 1) = \frac{n^2 - n}{2}$$

A more subtle approach, however, leads to an algorithm which is $O(n \cdot \log_2 n)$. This approach is illustrated in Figure 7.13 for an array of 16 numbers involving the integers 1 through 16.

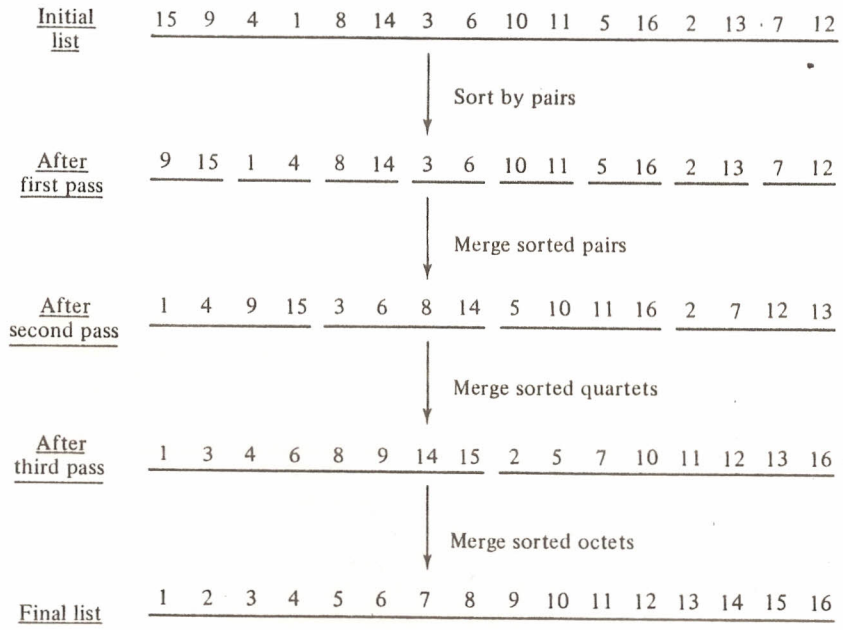

FIGURE 7.13 Illustration of an efficient sorting algorithm.

To describe this approach, let us assume, at first, that it so happens that $n = 2^k$, where k ($= \log_2 n$) is a positive integer. The algorithm then begins by forming a sorted pair of numbers after comparing the first and the second numbers in the initial list, a second sorted pair from the third and the fourth numbers, a third pair from the fifth and the sixth, and so on—see second row of Figure 7.13. The algorithm then goes through $k - 1$ more merging passes over the list, forming in the process sorted sublists of numbers, first, with four members, then with eight, then with sixteen, until finally, the whole list of numbers is fully sorted. The key observation that leads to the algorithm is that the merging of two *previously sorted* lists of length m requires at most $2m - 1$ comparisons [i.e., this type of merging is a $O(m)$ procedure]. Omitting

the details, it is clear that since there are $O(n)$ comparisons on each pass through the list of numbers and since there are k passes until the list is completely sorted (e.g., for $n = 16$ there are $\log_2 16 = 4$ passes), the algorithm is $O(nk) = O(n \cdot \log_2 n)$.

Exercise 7.4 Show in detail that the algorithm described above is $O(n \cdot \log_2 n)$.

What if the length of the list of numbers to be sorted, n, is not a power of 2? Let then $k = \lceil \log_2 n \rceil$ (where again the notation $\lceil a \rceil$ stands for "the smallest integer that is greater than or equal to a"). We may then add $2^k - n$ very large numbers to our initial list. After the expanded list has been sorted, its first n elements will constitute a sorted list of the original n numbers. The time required by our algorithm to sort the expanded list is $O(k \cdot 2^k)$, and since n is of the same order as 2^k and $\log_2 n$ differs from k by less than 1 unit, this is equivalent to $O(n \log_2 n)$.

The technique outlined above, which is typical of a common approach to many combinatorial problems, is often referred to as "divide and conquer." The idea, of course, is to solve the problem in small parts (e.g., by sorting initially only pairs of numbers as we did in the algorithm described above) and then to combine these parts in an efficient manner to arrive at the solution of the whole problem.

Another example of this approach solves efficiently the problem of adding a new number (at its proper place on the list) to an already sorted list of n numbers. By determining, first, whether the new number should be in the upper or lower half of the list [this can be done by comparing the new number with the $(n/2)$th element in the list], then the quarter in which it falls, and so on, this problem can be answered in $O(\log_2 n)$ time. Note that this procedure is equivalent to one that we would have followed if we were *searching* for a specific number on the list. Therefore, the procedure is known as an efficient algorithm for *searching* through a sorted list and, as we just saw, it is $O(\log_2 n)$.

7.2.2 Simulating the Locations of Urban Events

We now turn to what is perhaps the most fundamental geometrical question in the simulation of urban service systems: How do we simulate the locations of incidents, requests for service, or other events of interest which are distributed according to some prespecified probability law?

Suppose that the city (or part of the city) which is to be simulated is partitioned into small-area *reporting zones*, such as census blocks or police reporting areas or voting districts. If each reporting zone is small enough to be considered homogeneous, it would not be unreasonable to assume that

locations of events *within each zone* are uniformly distributed. The concept of the zone is identical to the concept of the atom (Chapter 5) or of the demand-generating node (Chapter 6). We choose to use the term "zone" here because it conveys better the idea of geographical entities of varying areas and shapes with which we are now dealing.

To obtain a sample location from within the city, the following two-step procedure can be used:

STEP 1: To decide in which zone the event location will be, obtain a sample from the probability mass function depicting the relative likelihood of events among zones.

STEP 2: To identify the exact location of the event, obtain a sample from a uniform distribution over the zone selected.

The second step is by no means trivial if we allow the zones to have arbitrary shapes—as we should to allow for maximum flexibility. An approach that works well in this respect is the use of the rejection method of sampling (Section 7.1) to generate sample values for two random variables simultaneously. We assume that any given zone can be modeled as a polygon with n sides (n arbitrary) having no special properties such as convexity.[5] Suppose that we enclose that polygon in the smallest rectangle fully containing it, with the sides of the rectangle constrained to be parallel to some prespecified set of coordinate axes (Figure 7.14). Then, using two random

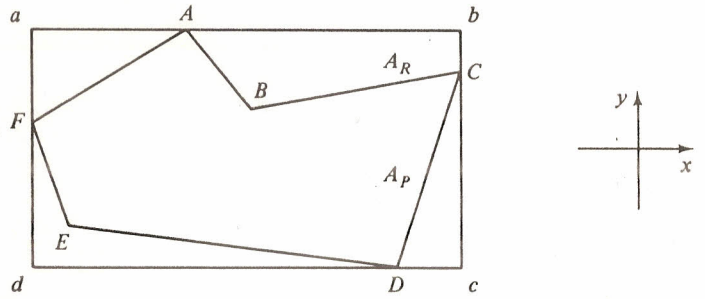

FIGURE 7.14 Polygon *ABCDEF* represents a reporting zone. It is inscribed in a rectangle *abcd* whose sides are parallel to the *x* and *y* axes.

numbers, a candidate point that has a uniform distribution over the rectangle is obtained. If this point is also within the polygon, it is accepted as the sample value; otherwise, it is rejected and new points are generated until one is accepted. The probability that any candidate point will be accepted is equal to the ratio of the area of the polygon (A_P) to the area of the rectangle

[5] This is a reasonable assumption, since we can approximate, to any desirable degree of accuracy, zones of any shape as polygons.

(A_R). The number of candidate points that have to be generated until one is accepted is a geometrically distributed random variable with mean A_R/A_P. For reasonably compact polygons, this number, reflecting sampling efficiency, is usually less than 2 (and often quite close to 1).

How does the computer determine whether a particular candidate point in the rectangle is inside or outside the polygon of interest? An effective way to deal with this problem is the point-polygon method.

Point-polygon method. The *point-polygon method* answers the following question: "Given a point (x, y) and a polygon specified by the n clockwise-ordered vertices $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$, is the point (x, y) contained within the polygon?" The basic idea of the method is to extend a ray in any direction from the point in question; if the ray intersects the sides of the polygon an *odd* (*even*) number of times, the point *is* (*is not*) within the polygon [NORD 62, BROW 68]. Figure 7.15 illustrates this idea. The method "works" because if the point is outside the polygon, the ray will exit the polygon each time that it enters it, since it must end up outside the polygon; the number of exits equaling the number of entries implies that the total number of intersections must be an even number. If, on the other hand, the point is within the polygon, then the number of exits e must be one greater than the number of entrances $(e - 1)$, because the initial exit is not paired with an entrance; the total number of intersections $2e - 1$ is clearly an odd number.

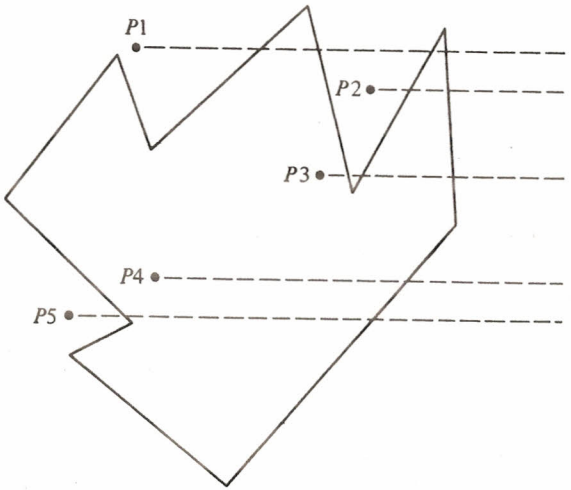

FIGURE 7.15 Point-polygon method. Rays from P_1, P_2, and P_5 interesct the polygon an even number of times and thus P_1, P_2 and P_5 are *not* within the polygon. Rays from P_3 and P_4 intersect the polygon an odd number of times and thus P_3 and P_4 are within the polygon.

The method is completely general and does not require any special properties of the polygon. It is particularly well suited for machine implementation, since the tests for intersection are quickly performed on a computer (see also Problem 7.4).

7.2.3 Identifying the Zones Where Events Occur

We have just seen how to generate events that are geographically distributed among and within zones in a city in accordance with some prespecified probability law. The reverse question is the following: "Given a point (x, y), in which geographical zone is it contained?"

Clearly, the point-polygon method is also helpful in answering this question: it can be applied to each zone in turn, until a positive result is obtained. If the polygons to be tested, however, are many and have a relatively large number of sides, the computer time for this search will be excessive.

The search can be made more efficient by again embedding each reporting zone in the smallest rectangle that contains it, just as before (Figure 7.16). For any particular zone, denote the respective vertices of this rectangle as follows: $(x_{\min}, y_{\min})$, $(x_{\max}, y_{\min})$, $(x_{\max}, y_{\max})$, $(x_{\min}, y_{\max})$. For the point (x, y) to be within the zone, the following inequalities must be satisfied:

$$x \leq x_{\max} \tag{7.18a}$$

$$y \leq y_{\max} \tag{7.18b}$$

$$x \geq x_{\min} \tag{7.18c}$$

$$y \geq y_{\min} \tag{7.18d}$$

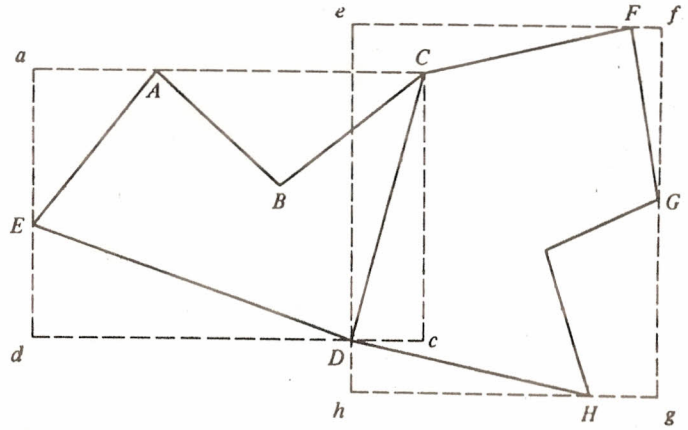

FIGURE 7.16 Rectangles *aCcd* and *efgh*, which contain the contiguous zones *ABCDE* and *CFGHD* overlap.

For large cities, the test indicated by inequality (7.18a) fails about 50 percent of the time, causing immediate rejection of the zone in question. If inequality (7.18a) is satisfied, inequality (7.18b) also has a conditional rejection rate of approximately 50 percent. The complete point-polygon procedure need be carried out only for those zones which satisfy inequalities (7.18), thus greatly decreasing the search time spent on obvious bad choices. It should be noted, in any event, that more than one candidate zone may satisfy inequalities (7.18), owing to the likely overlaps of the zone-containing rectangles as shown in Figure 7.16.

Identifying the zone in which a point (x, y) is contained is likely to be a task that must be repeated a very large number of times in the course of a simulation. In that case, the procedure described above can be speeded up further by creating sorted lists, one each for the x_{max}, y_{max}, x_{min}, and y_{min} coordinates of the vertices of the zone-containing rectangles. This can be done by using our efficient divide-and-conquer sorting algorithm. The identification of the rectangles (and zones) that satisfy inequalities (7.18) can then be accomplished quickly from the sorted lists, through use of our efficient searching algorithm.

7.2.4 Do *n* Line Segments Intersect?

Suppose now that we are given n straight-line segments on a plane with each segment specified by its two end points. An interesting question then is: Do *any* of these line segments intersect?

The straightforward approach requires $O(n^2)$ time: there are $n(n-1)/2$ possible pairs of line segments which must be checked for intersections, and checking for the intersection of any given pair of line segments requires constant time.

Once again, however, the most straightforward approach is not the most efficient one. We describe next an algorithm that requires $O(n \log_2 n)$ time [SHAM 76] to answer the problem: "Is there at least one intersection? If so, identify one."

In the x–y coordinate system we shall call the *left* (*right*) *end point* of a given line segment that end point of the segment with the smaller (larger) value of the x-coordinate.[6] Suppose now that we begin "sweeping" a vertical line along the x-axis (Figure 7.17). Two line segments S and T are then said to be *comparable* at $x = x_1$ if they are both intersected by the vertical line at $x = x_1$. The segments are *consecutive* if S is *immediately* above or *immediately* below T; in the former case we write $S =$ above (T, x_1), in the latter $S =$ below (T, x_1). Note that it is possible that no line segment will be above and/or below a line segment T for some or all values of x.

The key observation leading to the algorithm below is that if two line

[6] If a line segment happens to be vertical, we can adopt any convenient convention for distinguishing between a left and a right end point.

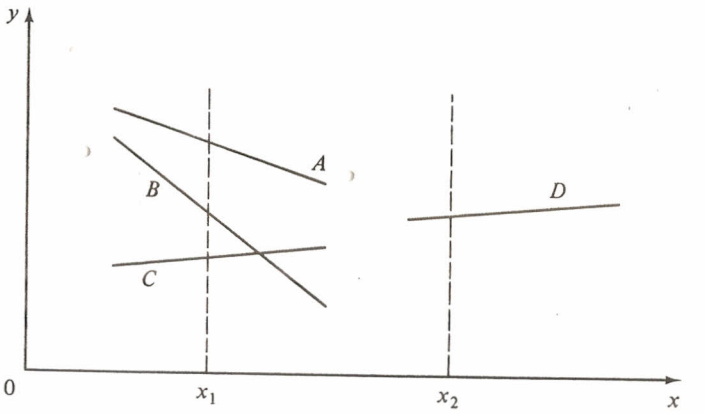

FIGURE 7.17 Illustration of relations between line segments. A = above (B, x_1), C = below (B, x_1), ϕ = above (D, x_2).

segments S and T intersect for some value of $x = x_1$, they must be consecutive[7] for some range of $x < x_1$. The algorithm thus concentrates on those line segments which become consecutive for some range of values of x.

The algorithm begins by sorting the $2n$ end points from left to right according to the values of their x-coordinates and then sweeps through the $2n$ end points from left to right. During the sweep, the algorithm maintains a list of *current line segments* which is sorted according to the order in which the line segments intersect the vertical sweep line. This sorted list is updated every time an end point is encountered: if the end point is a left end point, the corresponding line segment is added to the sorted list at the appropriate place; if it is a right end point, it is deleted from the list (see also Figure 7.18). The algorithm performs a check on whether two line segments intersect only when they become consecutive in the sorted list of line segments. It stops and returns the names of two intersecting line segments as soon as it finds an intersection.

We can now present the details of the algorithm:

Algorithm for the Intersection of Line Segments (Algorithm 7.1)

STEP 1: Sort the x-coordinates of the $2n$ end points of the line segments from left to right. Begin processing the $2n$ end points at the leftmost end point.

STEP 2: Let x_i denote the x-coordinate of the end point currently being processed and X the line segment corresponding to that end point.

[7] With the exception of special cases in which 3 or more line segments intersect at the same point.

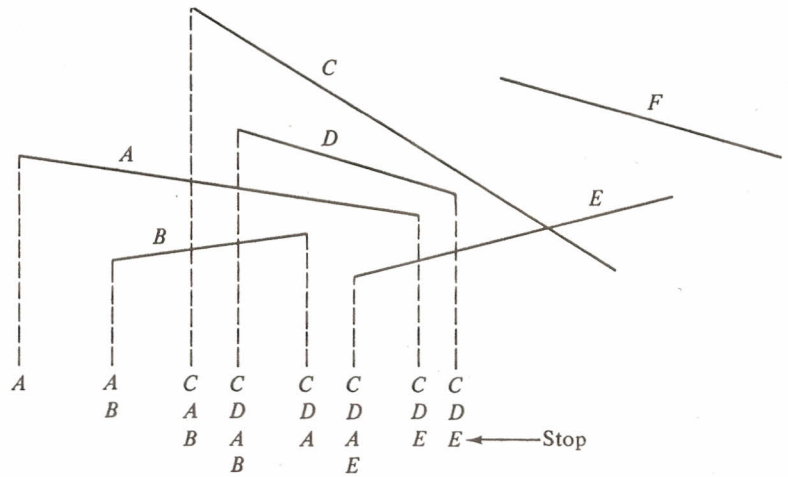

FIGURE 7.18 Illustration of Algorithm 7.1. The sorted list of current line segments is shown at all end points processed by the algorithm. The algorithm stops at the point indicated, since it discovers there the intersection of C with E.

1. If this is the left end point of X:
   a. Add X to the sorted list of current line segments, at its appropriate place according to the y-coordinate of the end point.
   b. If Y = above (X, x_i), check to determine if Y and X intersect. If they do, *stop* and return "Y and X." If they do not, continue.
   c. If Z = below (X, x_i), check to determine if Z and X intersect. If they do, *stop* and return "Z and X." If they do not, go to Step 3.
2. If this is the right end point of X:
   If Y = above (X, x_i) and Z = below (X, x_i), check whether Y and Z intersect. If they do, *stop* and return "Y and Z." If they do not, delete X from the list of current line segments.

STEP 3: If the last end point processed was the rightmost end point, *stop* and return "no intersection." If not, go to the next end point (to the right) and return to Step 2.

To prove that this algorithm works, (i.e., that it finds an intersection if there is at least one), it is sufficient to show (see Problem 7.5) that *if the algorithm is not stopped when it finds its first intersection*, it is guaranteed to find eventually the leftmost intersection of the line segments. This statement

emphasizes the fact that the first intersection discovered by Algorithm 7.1 is not necessarily the leftmost intersection. This peculiarity is illustrated in Figure 7.19. Another peculiarity of Algorithm 7.1 is that, if allowed to process all end points, it may "miss" some intersections (see again Figure 7.19), although, as we just stated, it will *never* miss the leftmost one.

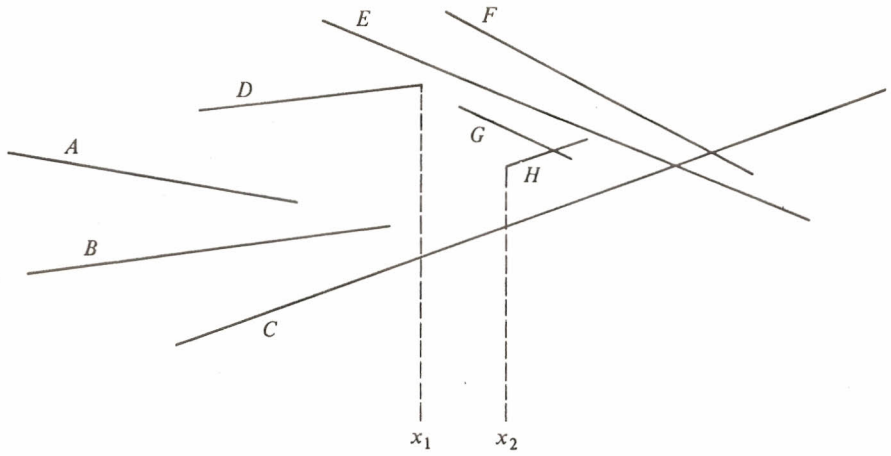

FIGURE 7.19 Algorithm 7.1 will detect first the intersection of C with E at the point x_1. Note that the intersection of C with E is not the leftmost one. If the algorithm is allowed to proceed, it will detect at x_2 the intersection of G with H. Algorithm 7.1 cannot detect the intersection of F with C (even if allowed to proceed).

What is the computational complexity of Algorithm 7.1? Step 1 requires $O(n \log_2 n)$ time for sorting the $2n$ end points. [It actually requires $O(2n \log 2n)$, but remember that in the theory of computational complexity $O(2n)$ is equivalent to $O(n)$.] The addition (or deletion) of line segments to (or from) the sorted list of current line segments in Step 2 requires $O(\log_2 n)$ time, since the sorted list can contain at most n line segments at any time. Checking whether two line segments intersect requires constant time (i.e., is independent of n). Since in the worse case, when there is no intersection, Step 2 must be repeated $2n$ times, Step 2 also consumes a total time which is $O(n \log_2 n)$. Noting that Step 3 is trivial and will be repeated $2n$ times in the worst case, we thus conclude that Algorithm 7.1 is $O(n \log_2 n)$, as suggested initially.

7.2.5 Intersections of Reporting Zones

We can now turn our attention to some of the more difficult questions that we have asked in the beginning of this section, such as: "Which ambulance zones have areas in common with a particular police precinct?" In general, we shall deal next with questions involving intersections of polygons.

From our earlier assumption that all elementary reporting zones in a city can be represented as polygons (with arbitrary numbers of sides) it follows that such entities as "districts," "precincts," "ZIP-code areas," and so on, which are combinations of one or more reporting zones, can also be represented as polygons (with no special properties such as convexity). The efficient "intersections algorithm" (Algorithm 7.1) which we discussed in Section 7.2.4 and the point-polygon method will be particularly useful in answering questions related to polygons.

It will be assumed below that all geographical entities under consideration are *simple* polygons. A simple polygon is one that has no intersecting sides (see Figure 7.20a). It is easy to check whether this condition is satisfied by using Algorithm 7.1. For an n-sided polygon, each side is viewed as a line segment and Algorithm 7.1 is applied. If no intersections are discovered, the polygon is simple. If it is not, the polygon can be viewed as a chain of two or more simple polygons, with neighboring polygons in the chain having only a finite number of corner points in common (Figure 7.20b). Each simple polygon in the chain can then be treated separately. Thus, the assumption that all geographical entities are simple polygons is not restrictive.

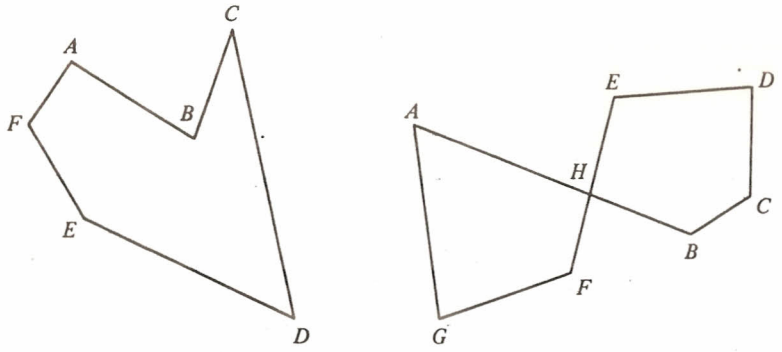

FIGURE 7.20 a) Simple polygon. b) Polygon *ABCDEFG*, which is not simple, can be viewed as a chain of two simple polygons *AHFG* and *EDCBH* with a common point at *H*.

An important issue now is deciding whether two simple polygons A and B overlap.[8] For this to happen, either some sides of A must intersect some sides of B, or A must be wholly contained in B (or vice versa). The following procedure then suggests itself. We shall assume, without loss of generality, that polygons A and B are both n-sided.

[8]We shall adopt the convention that if two polygons have only corner points and (possibly) sides in common, they *do not* overlap. Clearly, this convention could be modified, when necessary.

STEP 1: Treat the $2n$ sides of the two polygons as $2n$ line segments and apply Algorithm 7.1. If an intersection is discovered,[9] the two polygons must overlap, since each polygon is simple (and thus its n sides do not intersect). If no intersections are discovered, proceed to Step 2.

STEP 2: Take any corner point of polygon A and, using the point-polygon method, decide whether that point is contained in polygon B. If it is, polygon A is also entirely contained in B (and thus the two overlap). If it is not, take any corner point of B and, using the point-polygon method, decide whether that point (and thus B as well) is contained in A. If it is not, polygons A and B do not overlap.

Since in Step 1 we have $2n$ line segments, and thus $4n$ end points, the time required is $O(n \log_2 n)$. The point-polygon method (and Step 2) is $O(n)$. It follows that it is possible to decide whether two n-sided polygons (or reporting zones, districts, etc.) overlap in $O(n \log_2 n)$ time.

We can now answer the question which we posed earlier. To find which ambulance zones have areas in common with a particular police precinct, we apply the procedure described above k times (where k is the number of candidate ambulance zones), once for each ambulance zone. If both the police precinct and the ambulance zones are n-sided polygons, this question can be answered in $O(kn \log_2 n)$ time.

So far we have been able to check whether zones overlap with each other and, if so, to identify the pairs of zones that overlap, as well as whether the overlap is partial or if one zone is fully included in another (and which one). It is also possible to identify the polygon that forms the intersection of two overlapping zones (i.e., to have the computer "draw" this intersection for us). Unfortunately, however, there are no shortcuts for doing this faster than the straightforward method: since, in the worst case, each side of polygon A will intersect every side of B (and vice versa), there will be a total of n^2 intersections (if A and B are n-sided polygons), and therefore the computational effort must also be at least $O(n^2)$—see also Figure 7.21. It should be noted that the boundary of the polygon that describes the intersection of two polygons A and B consists of a part of the boundary of polygon A, followed by a point where a side of A intersects with a side of B, followed by a part of the boundary of B, followed by an intersection point of the boundaries, followed by a part of the boundary of A, and so on (see Figure 7.22). This, of course, assumes that A is not fully contained in B, or vice versa.

[9]A special procedure must be followed whenever this intersection happens to coincide with a corner point of the polygons.

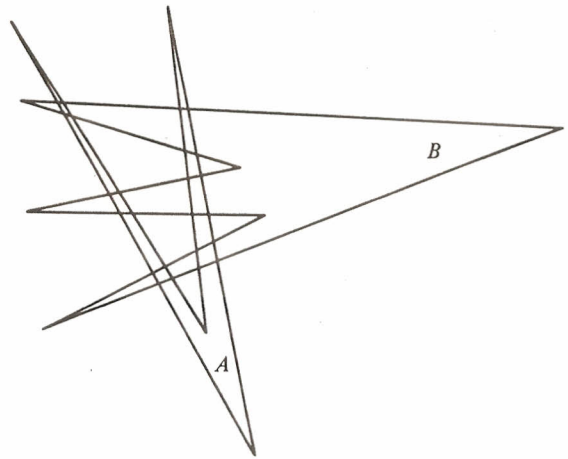

FIGURE 7.21 All sides of polygon *A* intersect with all sides of polygon *B* and form six disconnected areas in which *A* and *B* overlap.

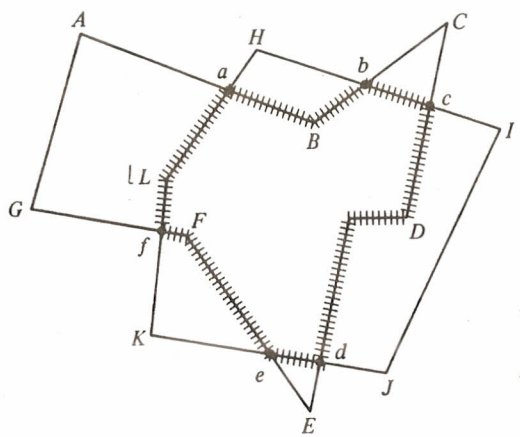

FIGURE 7.22 The intersection of polygons *ABCDEFG* and *HIJKL* is bounded *alternatively* by parts of the boundaries of the two polygons.

7.3 "ADVANCING THE CLOCK" IN A SIMULATION

Sections 7.1 and 7.2 have discussed two sets of theoretical problems that arise in simulations of urban services. In this and the next section we shall turn to two questions that always arise in connection with the actual preparation of simulation programs. We shall first address the problem of "advancing the clock" in a simulation.

In terms of simulating the passage of time, there are two basic types of simulations: (1) fixed-time-increment simulations, and (2) event-paced simulations. In fixed-time-increment simulations, time is advanced by regular intervals, say, every 8 seconds or every hour or every 3 years. The length of the fixed interval is, naturally, predetermined by the programmer and is chosen on the basis of the context of the simulation. In a macroscopic simulation of the future of the economy of a large city, for instance, the fixed time increment might reasonably be chosen as 1 year. The simulation would then follow from year to year the changes in the various economic activities in that city. On the other hand, a simulation concerned with producing estimates of the average response time by a local police department to emergency calls might use a fixed time interval of, say, 1 minute or of 30 seconds or less—depending on the amount of accuracy desired. In general, the length of these fixed time intervals are of the same order of magnitude as what might be called loosely the "natural time units" of the simulated system.

The concept of an event-paced simulation is less obvious than that of fixed time increments. It is an extremely useful one, however, and is used at least as often as the fixed-time-increment approach—especially in the case of the simulation of urban service systems. Basically in this case, the clock is advanced only to those instants in time when certain important (for the simulated system) *events* take place. Such events might be the arrival or departure of customers in a queueing system, or the occurrence of a fire alarm in the simulation of a fire department. Obviously, in event-paced simulations, time is advanced in irregular intervals.

No general guidelines can be offered regarding the choice between fixed-time-increment and event-paced simulation. This choice depends on the requirements of the problem. Clearly, however, an event-paced simulation provides a maximum amount of detail about activities during the simulated period and does not waste any effort dealing with times at which "nothing occurs."

7.3.1 Event-Paced Simulation of a Spatially Distributed Queueing System

We now present an example that illustrates the event-paced method for advancing the clock as well as some ideas that are often used in the simulation of queueing systems.

The specific example that we shall use is identical to the emergency repair problem described in Section 5.2, with one exception: it will be assumed that, whenever there is a queue of calls waiting for service at a time when the repair unit is busy serving a call at point (x, y), the next call to be selected for service is the one nearest to (x, y). (If there is no queue of calls, the unit is assumed to remain stationary at the location of the last call that was served and wait for the next call.) We shall define "nearest" in terms of travel time.

To be specific, let us assume that the service region in this case is a 2- by 1-mile rectangle (Figure 7.23); a right-angle metric is in effect, with directions of travel parallel to the sides of the region; the effective travel speed is constant at 20 mph; calls for service are generated in a Poisson manner at the rate of three per hour; the locations of calls for service are independent and uniformly distributed over the service region; and the amount of time on the scene needed for the emergency repair has the pdf shown in Figure 7.24. Times on the scene needed for successive calls are statistically independent.

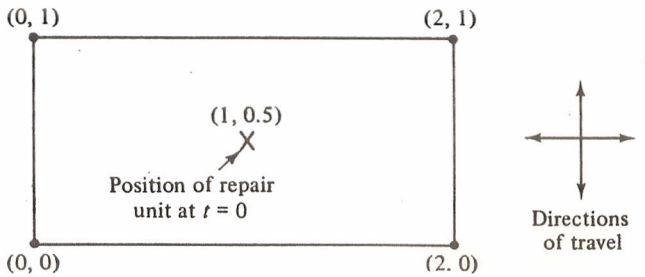

FIGURE 7.23 Service region of interest.

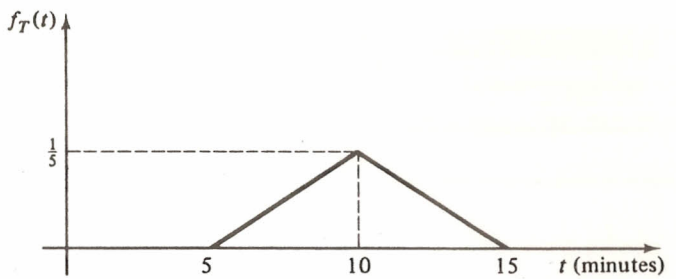

FIGURE 7.24 The PDF for time on the scene.

The total service time to a call consists of the sum of travel time to the location of the call plus time on the scene. Despite the independence of successive on-the-scene times, successive total service times are *not* independent, because of the "nearest-point" criterion of priority. To understand why, observe that, after a call which requires a long service time, it is more likely that the repair unit will be dispatched to a relatively nearby point (owing to the increased probability of several spatially distributed calls in queue).

Because of this dependence between service times, this situation cannot be analyzed (exactly) through classical queueing theory models. We shall therefore use simulation.

Suppose that we wish to simulate this spatially distributed queueing system over a period of, say, 12 hours and to collect some statistics such as the average time between reception of a call and the instant when the unit arrives on the scene, the maximum observed queue length, and so on.

We shall assume that at time $t = 0$, there are no calls in the "system" and that the emergency repair unit is located at the center of the service region. [We shall use a x–y coordinate system with the origin (0, 0) at the southwest corner of the region. We shall also use "minutes" as our time units.]

A number of time variables must be defined before we can proceed to the simulation program. Let

t = variable indicating real time

t_{nc} = time of the reception of the next call ("next call")

t_{ss} = time when service to a call begins ("service start")

t_{sf} = time when service to a call ends ("service finish")

t_l = time when the simulation ends ("last" time) = 720 minutes

We also need two indicators:

I = number of calls in the queue

$$N = \begin{cases} 0, & \text{if the repair unit is idle} \\ 1, & \text{if the repair unit is busy} \end{cases}$$

It should be noted that service to a call begins as soon as the trip of the repair unit toward the location of that call begins.

We shall use the index j to denote calls in order of reception by the repair unit's dispatcher. Associated with call j is the time of its arrival, t_j, and the location of the call (x_j, y_j). The current location of the repair unit is denoted by (x_0, y_0). Then the travel time needed to travel at 20 mph to a call at (x_j, y_j) once the unit completes service at a location (x_0, y_0) is given *in minutes* by

$$\tau = 3(|x_j - x_0| + |y_j - y_0|) \tag{7.19}$$

Finally, we shall use two random-number generators, one for probabilistic quantities related to time and the other for the locations of the incidents. We shall use $r_{1k}(k = 1, 2, \ldots)$ and $r_{2m}(m = 1, 2, \ldots)$, respectively, to denote random numbers provided by the two generators. It is important to keep in mind that every appearance of r_{1k} or r_{2m} in a logical box of a flowchart implies that the computer must draw a *new* random number from the appropriate random-number generator each time this box is entered during the execution of the program.

The computer program that simulates the foregoing problem is now outlined in flowchart form in Figure 7.25. The program consists of an initialization step (upper left-hand box), a main program, and a number of subroutines (subprograms). The main program—the "scheduler"—is concerned with finding out which is the next "event" that takes place and then instructs the computer to execute a subroutine accordingly. The events of interest are

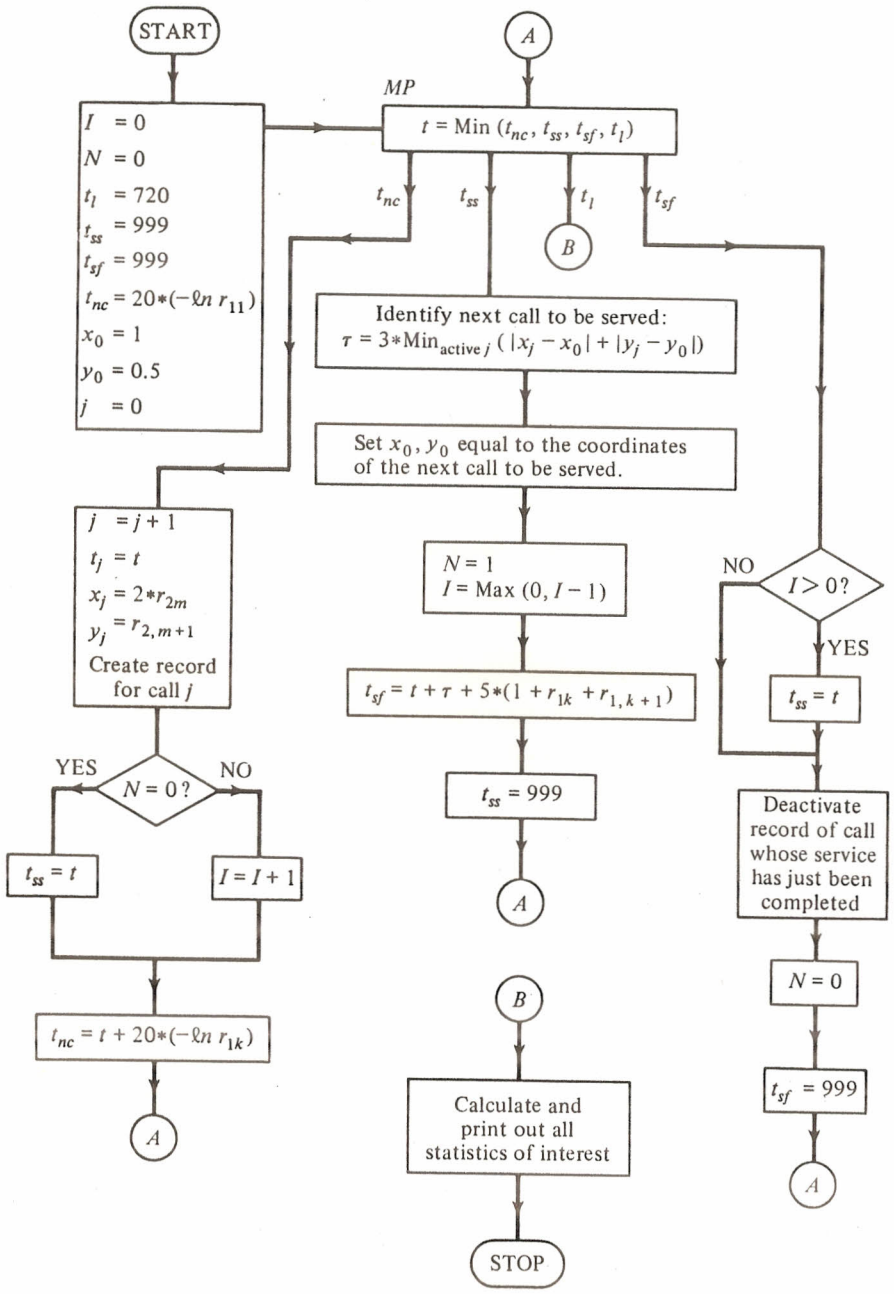

FIGURE 7.25 Flowchart for the simulation of a spatially distributed queueing system.

of four kinds: an arrival of a new call, the initiation of service to a call, the completion of service to a call, and the end of the simulation. The latter event occurs just once in the simulation.

Thus, the main program is just the logical box indicated as MP in Figure 7.25. Note that the actions that the computer takes upon the occurrence of each event are repetitive. For instance, following the arrival of a new call, at time $t = t_{nc}$, the computer first records the time and location of the call and then checks to find out whether the repair unit is idle or not. If it is idle ($N = 0$), it immediately assigns the newly arrived call to the unit ($t_{ss} = t$). If it is busy, the newly arrived call must join the existing queue and the length of the queue must, therefore, be increased by one ($I = I + 1$). In either case, the time of arrival of the next call is projected [$t_{nc} = t + 20(-\ell n\, r_{1k})$] and we return to the main program.

The reader should now review carefully the rest of the flowchart. It is particularly important to gain an understanding of how the time, t, "jumps" from event to event through the Min $(\cdot)$ operator in the MP box. Time intervals between these jumps are not equal, because events occur at unevenly spaced intervals. Note also how by having two random-number generators, we can control better our experiments (by repeating, for instance, the r_{2m} sequence that produces call locations while varying the r_{1k} sequence that controls the timing of events). We could, of course, have used more than two random-number generators.

How does the simulation end? Eventually, t_l(= 720 minutes) is bound to become the smallest of the variables t_{nc}, t_{ss}, t_{sf}, and t_l. This signifies that the simulation has run for the desired length of time and that we can now tabulate and print the desired statistics and then stop. This should also make clear to the reader the somewhat confusing reason for using the number 999, to which t_{ss} and t_{sf} are reset every once in a while. Obviously, there is nothing special about the value 999: any number greater than 720 would do. All that is required is a number large enough to keep t_{ss} and/or t_{sf} from ever being chosen as the minimum of the four variables when they are not supposed to be. We also make sure, in this way, that the simulation will eventually end.

Exercise 7.5 Explain the meaning of the equation

$$t_{sf} = t + \tau + 5(1 + r_{1k} + r_{1,k+1}) \tag{7.20}$$

in the t_{ss} subroutine. Note that this equation requires the generation of *two* independent random numbers.

We conclude this section by noting that the flowchart of Figure 7.25 can be easily modified to simulate a wide variety of other queueing situations—including much more complicated ones. We could, for instance, with the same basic logic simulate many cases involving classes of calls with priorities

assigned according to class of the call, queueing systems with finite queue sizes, spatially distributed queueing systems with time-dependent arrival and service rates, and so on. Extensions of this type are explored in Problems 7.8 and 7.9.

7.4 CHOICE OF A LANGUAGE FOR A SIMULATION

An important decision in designing a simulation is whether to use a general-purpose computer language (e.g., FORTRAN, PL/1, or BASIC) or a special-purpose simulation language (e.g., GPSS or SIMSCRIPT). Special-purpose simulation languages can simplify greatly the task of programming and carrying out simulation experiments—at some cost, however, to the flexibility of the program.

Two points need to be emphasized here. The first is that the two best-known special-purpose simulation languages, GPSS and SIMSCRIPT, are very well suited for simulating certain types of urban operations research problems. The reason is, essentially, that both of these languages (they could better be described as computer software systems) offer convenient features for simulating *probabilistic events* and are especially oriented to the analysis of *queueing networks*. Many problems in urban operations research can be viewed as queueing network problems, as we have seen. Such problems involve certain logical sequences that appear time and again (such as the arrival of a user at a facility, the waiting there until the facility becomes available, and, eventually, the service process at the facility). Rather than reprogramming these oft-repeated logical sequences from the beginning, special-purpose languages provide standardized codes for them. Similarly, the types of data and statistics that are collected during simulations of queueing systems are more or less the same every time (e.g., the average waiting time or the length of the queue at various time instants). Special-purpose languages compute and tabulate automatically many of these data and statistics with practically no effort required on the part of the programmer.

The second point to be emphasized is that general-purpose and special-purpose simulation languages are *not* mutually exclusive. SIMSCRIPT, in particular, is especially designed for the inclusion of segments of FORTRAN code in a simulation program. Although more cumbersome, GPSS programs can also be made compatible with other subroutines written in general-purpose languages. For example, in simulating a spatially distributed queueing system, one may wish to use, say, a FORTRAN code to establish the geographical relationships between incidents and servers, and subsequently use GPSS code to describe the queueing and service processes. Recommended references for GPSS and SIMSCRIPT (both of which have undergone numerous revisions and improvements over the years) include [GORD 75, KIVI 69, and SCHR 74].

7.5 ADVANTAGES, DISADVANTAGES, AND MISUSE OF SIMULATION

Simulation has its strengths and its weaknesses as a methodology: there are certain types of problems for which the use of simulation is highly appropriate, likely to produce useful results, and cost-effective; for other types of problems, this is not true. There is nothing unusual about this: no single analytical methodology can deal with all quantitative problems that one may encounter. Nonetheless, simulation has been the subject of an unusual amount of controversy over the last decade or so (when it truly came into its own as a widely used approach). This phenomenon can be attributed to the fact that inherent in the concept and the mechanics of a simulation model or experiment, there is the potential, to an exceptional degree, for *misuse* of the technique. In this section we discuss briefly some of these issues. Chapter 8 presents a more detailed discussion of the implementation and proper use of models, in general, in urban operations research.

7.5.1 Advantages of Simulation

In deciding whether simulation should be part of a project aimed at understanding, modifying, or designing the operation of an urban system, it is important to keep in mind what this technique can offer, over and beyond the analytical approaches described in Chapters 2–6. First, simulation can be a *method of last resort* for problems that are mathematically intractable by any other techniques. In Section 7.3 we described a simple problem with a single server and a (very natural) shortest-distance dispatching strategy for which there are no results from queueing theory, exact or otherwise—even under steady-state conditions. One does not have to think hard to identify a large number of similarly difficult but important problems. Second, even for problems that are mathematically tractable, simulation can often provide *a higher level of detail* than can other techniques. For instance, while queueing theory might provide good approximations to many aggregate (fleet-wide) statistics for a fleet of spatially distributed urban service vehicles, we might still wish to simulate that fleet in order to study, say, specific vehicles in specific parts of the city. Finally, simulation can sometimes provide (approximate) *answers at a lesser cost (or effort)* to some problems which are fully tractable mathematically but whose solution may be cumbersome and time-consuming. It may sometimes be easier, for example, to estimate the average distance between two random points in an area with a complicated shape through simulation (by generating many random pairs of points and "measuring" the distance between them) than through the geometrical probability techniques of Chapter 3.

One should also not lose track of the fact that simulation is based on *experiments*. In this respect the two main advantages of simulation, by com-

parison to experiments in the "real world," are that it *abbreviates time* immensely and that it makes it possible to perform what would in reality be *very expensive experiments with very expensive systems*. "Expensive" is used here not only in the sense of "costly" but also of "high risk," both in a physical and in a political respect. It is next to impossible for a fire department administrator, for example, to ever authorize drastic modifications in the allocation of fire companies in a borough as a temporary experimental measure. However, one can simulate these modifications and observe in a few seconds how, in an approximate way, the modified system would have reacted to a sequence of fire alarms that is an exact duplicate of all the fires that took place in a city in a whole year [IGNA 78]. Actual experiments with urban service systems are becoming increasingly unusual and costly (see, e.g., [KELL 74] for a description of a recent one). Moreover, unless such experiments are predesigned with meticulous care, they are bound to raise more questions than they answer [LARS 75]. Simulation packages, therefore, may soon be providing the only available preliminary testing ground for some of these experiments. In this respect, other advantages of simulation include that it *permits modification or design* of urban service systems by *trial and error*, allows for easy *exploration of the system's sensitivity* to changes in the input parameters, and provides *a highly controllable environment* for experiments (allowing duplication of probabilistic sequences of events through the repetition of random number sequences) to an extent unparalleled even in the traditional physics or chemistry laboratory.

Last but not least, simulation can be valuable to the urban operations researcher as a means of *testing the applicability and validity of mathematical models and expressions*. Throughout this book we have developed mathematical expressions for quantities of interest in the urban environment which were often derived from idealized models of what actually happens in practice. An excellent way to see how valid these expressions are is to compare their predictions with the results of more realistic and detailed simulation models. We have seen several instances [e.g., expressions (3.93) and (5.64)] where simple mathematical expressions that might have seemed to provide only the crudest approximations to reality were shown by simulation to provide highly accurate and reliable predictions [IGNA 78].

7.5.2 Disadvantages of Simulation

The disadvantages of simulation as a technique are also primarily due to the fact that simulation is basically an experimental (or empirical) approach to solving problems. As a result (and this is probably the most important disadvantage) it is difficult to *develop cause-and-effect relationships* through simulation, especially when the system under consideration requires the specification of many input parameters and involves complex interactions,

as is most often the case with urban systems. Despite the fact that one can conduct sensitivity analyses by varying systematically some (or all) of the simulation inputs, the range of values that can be tried is usually limited due to time or cost limitations. In view of this, it is dangerous to extrapolate and make predictions on the basis of whatever functional relationships between inputs and outputs one is able to develop through simulation.

A second disadvantage is that, like all empirical techniques, *simulation tends to be a relatively expensive mathematical tool*. It is almost impossible to find examples where the resources (in terms of manpower, money, and time) necessary to develop, validate, and run an urban simulation model which is *truly* useful to the analyst or to the agency for which it was developed were not seriously underestimated at the outset.

Finally, as with all experimental results (and, actually, more so) *the statistical analysis of simulation results is difficult*. What is the effect of the starting conditions of the simulation on the final results? How many data points should be disregarded as reflecting primarily the starting conditions and not the long-term characteristics of the simulated system? What is the statistical confidence that can be attached to the results? In the course of a parametric analysis, have we discovered a local or a global optimum set of operating conditions? These are all important questions which, because of space limitations, have not been addressed in this chapter. Good introductory treatments can be found in Naylor et al. [NAYL 69] and in Gross and Harris [GROS 74], and a more advanced treatment in Fishman [FISH 73, 78].

7.5.3 Misuses of Simulation

By "disadvantages" of simulation, we mean difficulties that are an inherent part of the technique. By "misuses" we mean problems that result from the naive or erroneous or—in the worst case—even dishonest uses of simulation. We discuss such misuses below.

By far the most common is the use of simulation as *an expensive way of obtaining obvious or easily derivable results*. The coauthors of this book have encountered numerous instances where large-scale simulation models have been developed and are being used (often at great cost in money[10] and manpower) to provide information that is readily obtainable otherwise, through some simple mathematical expression. The tendency for such misuse is particularly pronounced within government agencies, because of the fact that it is often easier in such cases to obtain a budgetary appropriation for computer-related expenditures (including computer programmers and "systems analysts") than for personnel with analytical competence and skills.

[10] The cost of using simulation models in such a thoughtless manner is often exacerbated by the temptation to "include everything" (the "kitchen sink") in a simulation, as noted in Chapter 8.

A second, potentially more damaging, misuse of simulation results from the fact that simulation packages are often used without any understanding of their logic and are mysterious "blackboxes" to all but the designers of the package (if that). For urban service systems, such packages may run into many hundreds, perhaps even thousands, of lines of computer code. Thus, it is practically impossible even for an expert to check fully the logic and assumptions of the simulation program, without assistance from its designer. As a result, a simulation may contain hidden (intentionally or unintentionally) assumptions which are unrealistic or only partly valid without its users being aware of the fact. The only way to avoid such unfortunate situations is by setting forth strict documentation requirements for the simulation's logic and for the program code (see Chapter 8), coupled with a continuing relationship between program developer and program user.

Finally, simulations can be misused by *placing too much faith* in their results. This is a problem that usually afflicts simulation users rather than those who develop the programs. The word "simulation" carries a connotation of "exact duplication" and "mimicking." It is often forgotten that simulations, despite including myriad details, are based on idealized models of reality that include many simplifying assumptions and approximations. Not unlike analytical models, it is the responsibility and obligation of the designer of a simulation program to warn its users constantly about the limitations of the simulation and thus prevent such misuse.

References

[BROW 68] BROWNSTEIN, S. J., "Some Concepts and Techniques for Constructing and Using a Geographically-Oriented Urban Data Base," *Socioeconomic Planning Sciences, 1*, 309–325 (1968).

[FISH 73] FISHMAN, G. S., *Concepts and Methods in Discrete Event Digital Simulation*, Wiley, New York, 1973.

[FISH 78] FISHMAN, G. S., *Principles of Discrete Event Simulation*, Wiley, New York, 1978.

[GORD 75] GORDON, G., *The Application of GPSS V to Discrete System Simulation*, Prentice-Hall, Englewood Cliffs, N.J., 1975.

[GROS 74] GROSS, D., AND C. M. HARRIS, *Fundamentals of Queueing Theory*, Wiley, New York, 1974.

[IGNA 78] IGNALL, E. J., P. KOLESAR, AND W. E. WALKER, "Using Simulation to Develop Analytic Models: Some Case Studies," *Operations Research, 26*, 237–253 (1978).

[KEEN 72] KEENEY, R. L., "A Method for Districting among Facilities," *Operations Research, 20*, 613–618 (1972).

[KELL 74] KELLING, G. L., T. PATE, D. DIECKMAN, AND C. E. BROWN, *The Kansas City Preventive Patrol Experiment: Summary Report*, The Police Foundation, Washington, D.C., 1974.

[KIVI 69] KIVIAT, P. J., R. VILLANUEVA, AND H. M. MARKOWITZ, *The SIMSCRIPT II Programming Language*, Prentice-Hall, Englewood Cliffs, N.J., 1969.

[LARS 75] LARSON, R. C., "What Happened to Patrol Operations in Kansas City? A Review of the Kansas City Preventive Patrol Experiment," *Journal of Criminal Justice, 3*, 267–297 (1975).

[NAYL 69] NAYLOR, T. J., *The Design of Computer Simulation Experiments*, Duke University Press, Durham, N.C., 1969.

[NEUM 51] VON NEUMANN, "Various Techniques Used in Connection with Random Digits," Paper 13 in *Monte Carlo Methods*, NBS Applied Mathematics Series 12, U.S. Government Printing Office, Washington, D.C., 1951.

[NORD 62] NORDBECK, S., *Location of Areal Data for Computer Processing*, Lund Studies in Geography, Series C, General and Mathematical Geography, No. 2, The Royal University of Lund, Sweden, C.W.K. Gleerup Publishers, Lund, Sweden, 1962.

[SCHR 74] SCHRIBER, T. J., *Simulation Using GPSS*, Wiley, New York, 1974.

[SHAM 76] SHAMOS, M. I., AND D. HOEY, "Geometric Intersection Problems," *Proceedings, 17th Annual Symposium on Foundations of Computer Science*, pp. 208–215 (1976).

Problems

7.1 *Hyperexponential random variable* Describe a method for generating sample values of a random variable X with a hyperexponential pdf

$$f_X(x) = \sum_{i=1}^{k} \alpha_i \lambda_i e^{-\lambda_i x} \qquad x \geq 0,\ \alpha_i \geq 0,\ \lambda_i \geq 0$$

where $\sum_{i=1}^{k} \alpha_i = 1$. (See also Problem 4.16.)

7.2 *Sample values of Gaussian random variables* Let Z be a Rayleigh random variable (cf. Chapter 3, Example 3) with pdf

$$f_Z(z) = \frac{z}{\sigma^2} e^{-z^2/2\sigma^2} \qquad z \geq 0$$

a. Show that a sample value, z, of A can be obtained by setting

$$z = \sigma\sqrt{-2 \ln r}$$

where r, as usual, denotes a random number ($0 \leq r \leq 1$).

In Chapter 3, Example 3, it was shown that if S and T are independent Gaussian random variables with zero mean and standard deviation equal to σ, that is, with pdf's

$$f_S(x) = f_T(x) = \frac{1}{\sqrt{2\pi}\,\sigma} e^{-x^2/2\sigma^2} \qquad -\infty < x < \infty$$

then the random variable $Z = \sqrt{S^2 + T^2}$ has the Rayleigh pdf shown above (the parameter σ is equal to the standard deviations of S and T).

b. Using this fact and the result of part (a), show that if X and Y are independent Gaussian random variables with mean m_x and m_y, respectively, and equal standard deviations, σ, then sample values, x and y, of X and Y can be obtained simultaneously by setting

$$x = m_x + \sigma\sqrt{-2 \ln r_1} \cdot \sin(2\pi r_2)$$
$$y = m_y + \sigma\sqrt{-2 \ln r_1} \cdot \cos(2\pi r_2)$$

where r_1 and r_2 are independent random numbers in the interval [0, 1]. This result has already been quoted [cf. expressions (7.13)–(7.15)].

Hint: Review carefully Example 3 of Chapter 3.

7.3 *Poisson demands with time-dependent rate, revisited* We have already seen a procedure for simulating the generation of demands in an urban service system according to a Poisson process with time-dependent mean rate $\lambda(t)$. That procedure (Section 7.1.4) required two "passes" over the interval [0, T]: once to determine the number of demands during [0, T] and a second time to generate the time instants when these demands occur. In this problem you will be asked to develop an alternative procedure which requires only a single pass.

Refer once more to Figure 7.12. Define a function $\lambda'(t)$ such that

$$\lambda'(t) \triangleq \lambda_0 - \lambda(t)$$

where λ_0 is the maximum value of $\lambda(t)$ in [0, T].

Suppose now that we used $t_i = (-\ell n\, r_i)/\lambda_0$ to generate sample values of successive demand interarrival times, beginning at $t = 0$. This, of course, would lead to too many demands being generated in [0, T]. (The expected number of demands would be equal to $\lambda_0 \cdot T$.)

How would you modify the foregoing procedure (by accepting or rejecting some demands) using $\lambda(t)$ and $\lambda'(t)$ to make it work correctly?

7.4 *Intersection of two line segments* Let the end points of the straight-line segment A be (x_1, y_1) and (x_2, y_2). Similarly, let the points (x_3, y_3) and (x_4, y_4) be the end points of the straight-line segment B. Devise a systematic procedure for checking whether A and B intersect.

7.5 *Further work with Algorithm 7.1* Refer to Algorithm 7.1 which examines a set of n line segments to find if at least one pair of them intersect.

a. Show that, if the algorithm is allowed to proceed after it discovers an intersection (if there are any), it will eventually discover the leftmost intersection in the set of line segments.

b. Modify the algorithm so that, if allowed to process all n line segments, it will find *all* existing intersections. (As stated, Algorithm 7.1 may miss some intersections.) What is the computational complexity of the modified algorithm?

7.6 *A systematic procedure for district partitioning* Consider a rectangular city district and two stationary facilities located at two arbitrary points (x_1, y_1) and (x_2, y_2) within the district. We wish to devise procedures that can be used by a computer to partition the district into two subdistricts so that all the points in the district are assigned to the facility closest to them. The "inputs" to the computer will be the coordinates of the locations of the two facilities and of the corners of the rectangle (the orientation of the rectangle with respect to the coordinate axes is arbitrary). The "outputs" are the coordinates of the corners of each subdistrict.

a. Devise a procedure for the case in which travel in the district is Euclidean.

b. Repeat part (a) for the case of right-angle travel, assuming that directions of travel are parallel to the coordinate axes. (In a special case in which parts of the boundary between the two subdistricts are not uniquely defined, use any arbitrary procedure you wish to specify that boundary.)

c. How would your procedure in parts (a) and (b) change if the district's shape was that of an arbitrary polygon with known coordinates for its corner points?

d. Assuming that a procedure has been devised for partitioning a district of arbitrary polygonal shape between *two* facilities (according to some arbitrary travel metric), how could that procedure be used to partition the district among n randomly located facilities according to the "closest facility" criterion?

Hint: Consider the case when a third facility is suddenly added in a district which has already been partitioned among two facilities. How would the old boundaries be modified?

Part (d) of this problem is also discussed in [KEEN 72].

7.7 *Simulating Buffon's needle experiment*[11] Recall Buffon's needle experiment, the famous early experiment involving geometrical probability concepts, from Section 3.3.1. Assuming that the length of the needle, l, does not exceed the spacing, d, between parallel lines ($l \leq d$), the probability that a randomly thrown needle intersects a line was shown to be

$$p = \frac{2l}{\pi d}$$

Thus, by conducting Buffon's experiment many times, one can obtain an estimate for π, say $\hat{\pi}$, through

$$\hat{\pi} = \frac{2l}{d}\left(\frac{\text{total number of needle tosses}}{\text{total number of intersections observed}}\right)$$

Suppose that we wish to perform these experiments by using computer simulation. After all, it may take a large amount of time to toss a needle, say, a few thousand times, although this has not been sufficient to discourage several people in the past from doing just that (up to 10,000 consecutive tosses have been reported).

Consider the following three alternative methods for simulating Buffon's needle experiment. (For each execution of the experiment, new random numbers must be drawn.)

Method 1

1. Obtain a random value for X, say x, from the uniform distribution on the interval $[0, d]$ (i.e., $x = r_1 d$).
2. Obtain a random value for S, say s, the sine of the angle α, which the needle forms with the set of parallel lines (see Figure 3.17) from the uniform distribution on $[0, 1]$ (i.e., $s = r_2$).
3. If $x \leq \frac{1}{2} ls$ or if $x \geq d - \frac{1}{2} ls$, an intersection has occurred.

Method 2

1. Obtain a random value y_1 for Y_1, from the uniform distribution on the interval $[0, d/2]$ (i.e., $y_1 = r_1 d/2$). Let $(0, y_1)$ represent the midpoint of the needle.

[11] C. W. Skinner, "Some Implications of Buffon's Needle Experiment for Simulation Modeling," ORSA/TIMS Joint National Meeting, Boston, April 1974.

2. Obtain a random value, x_2, for X_2, from the uniform distribution on the interval $[-l/2, +l/2]$ (i.e., $x_2 = r_2 l - l/2$).
3. Similarly, obtain a random value, y_2, for Y_2, using $y_2 = r_3 l - l/2$. $[(x_2, y_2 + y_1)$ will be the coordinates of some point lying on the needle.]
4. If $(x_2^2 + y_2^2)^{1/2} \leq \frac{1}{2} l$, go to Step 5. Otherwise, return to Step 2 and continue until a set of values (x_2, y_2) has been obtained that satisfies the foregoing test.
5. If

$$y_1 \leq \frac{l}{2} \sin \alpha = \frac{l}{2} \frac{|y_2|}{(x_2^2 + y_2^2)^{1/2}}$$

an intersection has occurred.

Method 3

1. Obtain a random value, x_1, for X_1 from the uniform distribution on the interval $[0, d]$ (i.e., $x_1 = r_1 d$).
2. Similarly, obtain a random value, y_1, for Y_1, using $y_1 = r_2 d$. Let (x_1, y_1) represent the midpoint of the needle.
3. Similarly, obtain random values (x_2, y_2), for (X_2, Y_2) using $x_2 = r_3 d$ and $y_2 = r_4 d$.
4. Let the end points of the needle lie on the line joining (x_1, y_1) with (x_2, y_2) at a distance $l/2$ from (x_1, y_1) in each direction.
5. If $y_1 \leq \frac{1}{2} l \sin \alpha$ or $y_1 \geq d - \frac{1}{2} l \sin \alpha$, where

$$\sin \alpha = \frac{|y_1 - y_2|}{[(x_1 - x_2)^2 + (y_1 - y_2)^2]^{1/2}}$$

an intersection has occurred.

a. Provide a physical interpretation of each one of the three methods by drawing figures that show schematically what each method does.

b. Which of the foregoing three methods are correct, which are incorrect, and why?

Hint: Two of the methods are incorrect.

7.8 *Queue with service priorities* Suppose that in the example of Section 7.3 we distinguished between two types of calls for police assistance: type 1 and type 2 calls. Type 1 calls enjoy nonpreemptive priority over type 2 calls. They also require constant service time, T_1, on the scene, whereas type 2 calls require constant service time, T_2, on the scene. Two-thirds of the calls on the average are type 2 calls and one-third are type 1. In all other respects, the operation of the single patrol car in this rectangular district is the same as described in Section 7.3.

Draw a flowchart similar to that of Figure 7.25 that describes how you would simulate this situation. You will note that the flowchart of Figure 7.25 contains all the essential ideas that you will need to simulate the new situation.

7.9 *Two police cars in district* Suppose that a second police car was added in the rectangular district described in the example of Section 7.3. Both cars are at the center of the district at time $t = 0$ and one of them is chosen arbitrarily to service the first call. From then on, whenever both cars are available at times when calls for assistance are received, the car closest to the location of the new call is dispatched there. When both cars are busy, arriving calls are placed in a queue and served according to the "shortest-travel-time" queue discipline (described in Section 7.3) as police cars become free. All problem parameters are exactly the same as in Section 7.3.

Draw a flowchart, at a level of detail similar to that of Figure 7.25, describing how you would simulate this situation. You will note that the flowchart of Figure 7.25 contains all the essential ideas that you will need to simulate the new situation.

8

Implementation

Any attempt to implement the models and methods of this book creates in itself a process that requires analysis. We would be poor analysts indeed if the extent of our reasoned thinking about an urban problem stopped with the mathematics. Rarely are we confronted with the simple situation in which all participants in a study share the same priorities and in which the modeling and data collection tasks are inexpensive and straightforward. More likely, things are much more complicated.

Since many readers of this book will eventually attempt to develop and utilize models in an actual urban setting, we wish to close the book by presenting some views on implementation. In the first part of the chapter, to illustrate the variety of forces and factors that can come into play, we provide thumbnail sketches of several "mini case studies" or "war stories" with which we are familiar. We assume that the reader has reviewed the eight steps in an operations research study that were discussed in Chapter 1. The case studies point to the difficulties of defining performance measures, prespecifying all constraints, identifying all decision makers, and retaining sustained commitment of key personnel. Motivated by the cases, in Section 8.2 we attempt to outline in a general way the model-related issues found to be important when attempting implementation. Here, for instance, experience has shown that a model's data-base requirement is often a much more troublesome factor than is model accuracy. Finally, in Section 8.3, we discuss the all-too-critical issues related to people and institutions that affect the success or failure of a model-based analysis. For example, a potentially fatal flaw in this area is for an analyst to work solely through a single individual in any agency—one who speaks "model language" and "agency language." Such an

Draw a flowchart similar to that of Figure 7.25 that describes how you would simulate this situation. You will note that the flowchart of Figure 7.25 contains all the essential ideas that you will need to simulate the new situation.

7.9 *Two police cars in district* Suppose that a second police car was added in the rectangular district described in the example of Section 7.3. Both cars are at the center of the district at time $t = 0$ and one of them is chosen arbitrarily to service the first call. From then on, whenever both cars are available at times when calls for assistance are received, the car closest to the location of the new call is dispatched there. When both cars are busy, arriving calls are placed in a queue and served according to the "shortest-travel-time" queue discipline (described in Section 7.3) as police cars become free. All problem parameters are exactly the same as in Section 7.3.

Draw a flowchart, at a level of detail similar to that of Figure 7.25, describing how you would simulate this situation. You will note that the flowchart of Figure 7.25 contains all the essential ideas that you will need to simulate the new situation.