S.G. Henderson and B.L. Nelson (Eds.), *Handbook in OR & MS, Vol. 13*

DOI: 10.1016/S0927-0507(06)13001-7

Chapter 1
Stochastic Computer Simulation

Shane G. Henderson

School of Operations Research and Industrial Engineering, Cornell University, USA
E-mail: sgh9@cornell.edu

Barry L. Nelson

Department of Industrial Engineering and Management Sciences, Northwestern University, USA
E-mail: nelsonb@northwestern.edu

Abstract

We introduce the topic of this book, explain what we mean by stochastic computer simulation and provide examples of application areas. We motivate the remaining chapters in the book through two in-depth examples. These examples also help clarify several concepts and techniques that are pervasive in simulation theory and practice.

1 Scope of the Handbook

What is "stochastic computer simulation?" Perhaps the most common example in everyday life is an electronic game, such as Solitaire or Yahtzee, that depends on a source of randomness to imitate shuffling cards, rolling dice, etc. The fidelity of the electronic game, which is a simulation of the physical game, depends on a faithful imitation of the physical source of randomness. The electronic game is useless (and no fun) otherwise. Of course, an electronic game usually needs a game player. If you replace the player by an algorithm that plays the game, and you compare different algorithms by playing many sessions of the game, then you have a pretty good representation of what stochastic computer simulation is and how it is used in operations research and the management sciences.

This book is a collection of chapters on key issues in the design and analysis of computer simulation experiments on models of stochastic systems. The chapters are tightly focused and written by experts in each area. For the purposes of this volume, "stochastic computer simulation" (henceforth just "stochastic simulation") refers to the analysis of stochastic processes through the generation

of sample paths (realizations) of the processes. We restrict attention to design and analysis issues, and do not address the equally important problems of representations (modeling) and execution (see, for instance, Fishwick, 1995). In broad terms, the goal of this volume is to survey the concepts, principles, tools and techniques that underlie the theory and practice of stochastic simulation design and analysis, emphasizing ideas and methods that are likely to remain an intrinsic part of the foundation of the field for the foreseeable future. The chapters provide an up-to-date reference for both the simulation researcher and the advanced simulation user, but they do not constitute an introductory level "how to" guide. (See, instead, Banks (1998), Banks et al. (2004), Law and Kelton (2000) or Fishman (2001). The latter book is at a slightly more advanced level than the others.)

Computer scientists, financial analysts, industrial engineers, management scientists, operations researchers and many other professionals use stochastic simulation to design, understand and improve communications, financial, manufacturing, logistics and service systems. A theme that runs throughout these diverse applications is the need to evaluate system performance in the face of uncertainty, including uncertainty in user load, interest rates, demand for product, availability of goods, cost of transportation and equipment failures. Much like the electronic game designer, stochastic simulation users develop models that (they hope) faithfully represent the sources of uncertainty in the systems that they simulate. Unlike the game designer, the simulation user also needs to *design the simulation experiment* – decide what cases to simulate, and how much simulation to do – and *analyze the results*.

Later in this chapter we provide two examples of the types of problems that are solved by stochastic simulation. The examples motivate and provide context for the remaining chapters in the book. We do not attempt – either in this chapter or in the remainder of the book – to cover the wide range of applications of simulation. A few important application areas are listed below.

Financial engineering/quantitative finance: The classical problem in this area is valuing a derivative, which is a financial instrument whose value depends on an underlying asset such as a stock or bond. Uncertainty in the value of the asset over time makes simulation a natural tool. Other problems in this vein include estimating the value-at-risk of a portfolio and designing hedging strategies to mitigate risk. See, for instance, Glasserman (2004).

Computer performance modeling: From the micro (chip) level to the macro (network) level, computer systems are subject to unpredictable loads and unexpected failures. Stochastic simulation is a key tool for designing and tuning computer systems, including establishing expected response times from a storage device, evaluating protocols for web servers, and testing the execution of real-time control instructions. See, for instance, Jain (1991).

Service industries: Service industries include call/contact centers, an application in which simulation has been used to design staffing and call-routing policies. Service applications emphasize delivering a specified level of service with a high probability; such issues arise in food delivery, financial

services, telecommunications and order fulfillment, for instance, and simulation helps to ensure quality service in the face of uncertainty.

Manufacturing: Stochastic simulation has seen extensive use in manufacturing applications. A few representative contributions include evaluation of production scheduling algorithms; work-center design and layout; estimation of cycle time-throughput curves; and evaluation of equipment replacement and maintenance policies.

Military: Military applications include life-cycle management of defense systems; combat and munitions logistics; crisis communications evaluation; and modeling network architectures for command and control.

Transportation and logistics: Simulation has been used to evaluate the effectiveness of Internet and cell-phone-based traveler information services; to perform benefits analyses for regional Intelligent Transportation Systems; and to design public transportation systems for the elderly and disabled. When the transportation system is part of a supply chain, simulation may be used to determine replenishment policies for manufacturers, distribution centers and retailers, or to estimate the impact of demand uncertainty on supplier inventories.

The *Proceedings* of the annual Winter Simulation Conference is an outstanding source of applications and success stories from these and other areas. The *Proceedings* from 1997 through the present can be found at *http://www.wintersim.org*.

The focus of this volume is narrow, by necessity, because the label "computer simulation" is attached to a number of activities that do not fall under the umbrella of "generating sample paths of stochastic processes". For instance, there is a vast literature on, and countless applications of, simulation of dynamic systems that are represented by differential and partial differential equations. Such systems may be stochastic, but the approach is to numerically integrate the system of equations through time to determine levels, probabilities, moments etc., rather than generating sample paths and averaging across repetitions as we do. Systems dynamics (see Sterman, 2000) is a popular approach for developing and analyzing differential equation models, and there is a substantial intersection between applications of systems dynamics and applications of stochastic simulation in operations research and the management sciences.

Although closely related, we do not consider issues that arise in person-in-the-loop simulations that are used for, among other things, training of personnel and evaluation of supervisory systems prior to insertion in the field. A key feature of stochastic simulation is that the source of randomness is under the control of the experimenter, which is not the case when a person is incorporated into the experiment. We also do not cover any type of computer animation, up to and including virtual reality, although this is certainly a kind of computer simulation. Stochastic simulation is sometimes a driving process for computer-generated animation, however. Nor do we consider the important area of parallel and distributed simulation; see Fujimoto (1999).

The next section introduces the key concepts we do cover via two examples.

2 Key concepts in stochastic simulation

There are a number of key simulation-related concepts that feature throughout this volume. The goal of this section is to introduce and explain those concepts through the use of two examples, and to link them to subsequent chapters. The reader familiar with stochastic simulation may still find it useful to briefly peruse this section to avoid potential confusion associated with slightly different uses of terminology. In preparing this section we have occasionally relied quite heavily on Nelson (2002, Chapter 4).

Discrete-event systems dominate simulation applications and so it is important to gain an understanding of simulation concepts related to such models. The processing-network example described in Section 2.1 is a special case. Many of the concepts introduced with this model extend to simulations of a broader class of models. The stochastic activity network example described in Section 2.2 reinforces some of the concepts introduced with the processing-network example, and brings out several new ones.

2.1 A processing network problem

A manufacturing system processes two classes of jobs with two processing stations (machines). Class 1 jobs are high-value jobs that need to be completed quickly. Class 2 jobs are of lower value, and while these jobs should be processed quickly, greater delays can be tolerated than with Class 1 jobs. Accordingly, the processing system has been set up as in Figure 1. Station 1 processes only Class 1 jobs. Station 2 is capable of processing both Class 1 and Class 2 jobs, so that it can assist Station 1 to complete the processing of Class 1 jobs. This network was first introduced and analyzed in Harrison (1996) and since then has become a standard example in the study of multiclass processing networks.

A *policy* is a strategy describing how the two stations coordinate their efforts in processing the incoming jobs. A key question is what policy should be used to ensure that Class 1 jobs are delayed for as short a period of time as possible, while ensuring that Class 2 jobs are also completed in a reasonable amount of time. A natural policy one might consider is for Station 1 to process jobs whenever they are available, and for Station 2 to give nonpreemptive priority to Class 1 jobs over Class 2 jobs. One might ask how this policy performs. It turns out that there is a rather surprising answer to this question. We will explain how simulation can be used both to develop an understanding of this system, and to explore a range of operating policies. To perform simulations of the model, we need to further specify its structure and decide how and what to simulate.

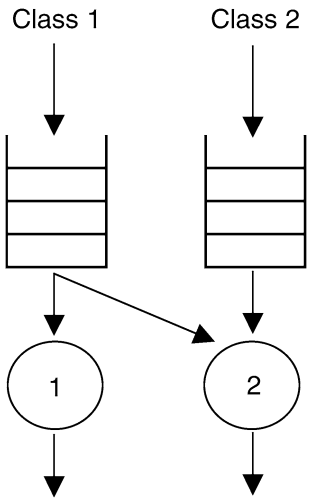

Fig. 1. Processing network.

A common approach for specifying simulation models is the *process-interaction view*. In this approach, one specifies the various entities that interact with a system and describes the processes they follow. In our example the entities are jobs of Class 1 and Class 2. Class 1 jobs arrive according to a certain arrival process and potentially wait in a queue of Class 1 jobs. Once they reach the head of the queue, they are served by either Station 1 or Station 2, whichever becomes available first. If both are available then they are served by Station 1. After service, Class 1 jobs depart the system. Class 2 jobs arrive according to a certain arrival process, potentially wait in a queue of Class 2 jobs, and then are served by Station 2 once the queue of Class 1 jobs is empty and Station 2 is available. They then depart the system.

This very natural description of the model helps to explain the popularity of the process-interaction view. Of course, the description of the model is incomplete, since we have not specified the arrival and service processes. Suppose that the times between arrivals of Class 1 jobs are i.i.d. random variables with distribution function F_{A1}. Suppose that, independent of Class 1 arrivals, the sequence of Class 2 interarrival times, the sequence of service times at Station 1, and the sequence of service times at Station 2 are mutually independent i.i.d. sequences of random variables with interevent distribution functions F_{A2}, F_{S1} and F_{S2}, respectively. Notice that Class 1 jobs and Class 2 jobs have the same service time distribution function F_{S2} at Station 2.

The sequence of interarrival times of Class 1 jobs is known as a *stationary* arrival process because all of the interarrival times have the same distribution. This is often a reasonable assumption in manufacturing models where considerable effort is exercised by managers to maintain a smooth flow of work through the system. However, in service systems where "jobs" are, in fact,

people, this assumption is often violated. Customer arrivals often exhibit significant "time-of-day" effects where arrivals are more frequent at certain times of the day. Such an arrival process is called "nonstationary", since the distribution of the time between customer arrivals can vary. Chapter 6 describes methods for modeling and simulating nonstationary processes. For now we maintain our assumption that the arrival and service processes consist of i.i.d. sequences of random variables.

The process-interaction view is one example of a *world view*, which is a general approach for specifying models for the purposes of simulation; see, e.g., Banks et al. (2004) and Law and Kelton (2000). The *event-scheduling world view* is another world view that closely mirrors the algorithm underlying most simulation software. We now give an event-scheduling view of the model that is couched in a discussion of how to simulate the model.

The key quantity to track is the vector $\mathbf{X} = (X_1, X_2, J_2)$ consisting of the number of Class 1 and Class 2 jobs in the system, and the class of the job in service at Station 2. If no job is in service at Station 2 then we set $J_2 = 0$. The vector $\mathbf{X}$ is known as the *system state* and it takes on values in the *state space*

$$\{0, 1, \ldots\} \times \{0, 1, \ldots\} \times \{0, 1, 2\}.$$

The state is modified by *events* which are arrivals of Class 1 or Class 2 jobs, and service completions at Station 1 or Station 2. These events occur at discrete epochs (points in time), and between event epochs the system state remains constant. Associated with each event is a *clock* that indicates the time that event is next scheduled to occur. The next event to occur is the one with the smallest clock reading. The simulation jumps immediately to the time of that event and then updates the system state and event clocks accordingly. For example, at the time of a Class 1 arrival, we first increment X_1 and schedule the next Class 1 arrival by generating a sample value from F_{A1}. Furthermore, if Station 1 is available, we start service on the job by setting the clock for service completion at Station 1 to the current time plus a sample from the distribution F_{S1}. If Station 1 is not available and Station 2 is, then we set $J_2 = 1$ and set the clock for a service completion at Station 2 to the current time plus a sample from F_{S2}.

At the time of any service-completion event we follow the logic dictated by the policy to determine which events to schedule next. For example, suppose that Station 2 completes a job of Class 2 (so that $J_2 = 2$). We then decrement X_2 and set J_2 to be either 0, 1 or 2 according to whether there are no jobs of either class waiting, there are Class 1 jobs waiting, or there are no Class 1 jobs waiting and one or more Class 2 jobs waiting. In the second or third case, the clock associated with service completions at Station 2 is set to the current time plus a sample from the distribution F_{S2}. When no job is in service at a station, there should be no service-completion event scheduled at that station. In that case we set the clock reading for the event to ∞. It then plays no role in the simulation until a job enters service at that station, at which time the clock is

reset. All that is needed now to set the simulation in motion is a specification of the initial system state and event-clock settings.

Notice that we need to be able to generate samples from a given distribution. This problem is almost invariably solved in two steps. In the first step we generate a sequence of numbers that mimics many of the properties of an i.i.d. sequence $U = (U(n)\colon\ n \geqslant 0)$ of uniformly distributed random numbers on $(0, 1)$; see Chapter 3. We then generate the desired samples assuming the existence of the sequence U using a variety of methods. See the remaining Chapters 4–7 in Part II.

The event-scheduling view of the model is perhaps not as intuitive as the process-interaction view. As noted in Nelson (2002, Section 4.7) the event-scheduling approach "…*defines* system events by describing what happens to the system as it encounters entities. The *process view* of stochastic processes *implies* system events by describing what happens to an entity as it encounters the system". A modeler is often able to visualize the progress of an entity through a system, and so the process-interaction view can be more intuitive than the event-scheduling world view.

The event-scheduling view is very similar to the algorithm used to simulate our model in most discrete-event simulation software. The discrete-event simulation algorithm for a general model can be approximately described as follows.

1. Initialization: Set the simulation clock T to 0. Choose the initial system state X and event clock readings $\{C_i\}$ say.
2. Let $T = \min_i C_i$ be advanced to the time of the next event and let I be the index of the clock reading that achieves this minimum.
3. Execute the logic associated with event I, including updating the system state X and event clocks $\{C_i\}$.
4. Go to Step 2.

Notice that even this simple description of the algorithm embodies several key issues.

- The system is tracked through a system state and clocks indicating the time at which events of different types are scheduled to occur.
- The simulation proceeds from event epoch to event epoch, rather than continuously in time.
- At event epochs the system state and clocks are updated, and any new events are scheduled by generating samples from appropriate distributions.

Our description of discrete-event simulation is somewhat specialized and imprecise, since to give a more general and precise definition would take us too far afield. The interested reader can find more general and precise descriptions in, for example, Shedler (1993), Glynn (1989) and Haas (1999).

The interarrival and service time sequences defined thus far are all *input* stochastic processes, in that they are specified by the simulation user. The logic

of the simulation model combines these input stochastic processes to form *output* stochastic processes. Output processes then shed light on the behavior of the system under study. Even for the simple model given above there are a host of stochastic processes one might consider. There are three processes that deserve special attention.

- Let $T(0) = 0$ and let $T(n)$ denote the time of the nth event epoch, $n \geqslant 1$. This value can be read from the simulation clock T just after Step 2 of the algorithm is completed. We call $(T(n)\colon\ n \geqslant 0)$ the *event-epoch* process.
- Let $X(0)$ be the initial system state, and let $X(n)$ be the system state immediately after the nth event epoch. Notice that $X(n)$ is the value of the system state immediately after Step 3 of the algorithm is completed. We call $(X(n)\colon\ n \geqslant 0)$ the *state-change* process.
- Let $Y(t)$ denote the state of the system at simulated-time t, and let $(Y(t)\colon\ t \geqslant 0)$ be the *state* process. We can recover this process from the event-epoch and state-change processes via $Y(t) = X(n)$, where n is chosen so that $T(n) \leqslant t < T(n+1)$.

The state process gives a continuous view of the system state with time. In our problem we are also very interested in the waiting times in queue of the two classes of jobs. In particular, we might also be interested in the stochastic process $(W_1(n)\colon\ n \geqslant 1)$, where $W_1(n)$ is the waiting time in queue of the nth Class 1 job to arrive to the system. It is difficult to construct this stochastic process from the event-epoch and state-change processes. Perhaps the easiest way to do this is to "tag" each Class 1 job as it arrives with the (simulated) time of its arrival. When the job enters service one can then subtract the job's arrival time from the current simulated time to give the waiting time in the queue. It is therefore straightforward to recover the waiting time sequence. These sorts of calculations are usually performed automatically by discrete-event simulation software.

Many other stochastic processes can be constructed in a similar fashion. The point behind constructing such stochastic processes is to enable the estimation of various *performance measures* that describe some aspect of the simulated system. A performance measure is essentially a summary statistic related to one or more of the stochastic processes described above. The choice of performance measure is invariably related to the questions one has in mind when a simulation model is built. In our example it makes sense to use measures related to the waiting times of jobs, but there are many possible performance measures. For example, we may be interested in any or all of the following.

PM1. For some fixed $t > 0$ let

$$\overline{Y}_1(t) = \frac{1}{t}\int_0^t Y_1(s)\,\mathrm{d}s$$

be the average number of Class 1 jobs in the system over the first t units of simulated time, where $Y_1(s)$ is the number of Class 1 jobs in the system

at time s. Notice that $\overline{Y}_1(t)$ is a random variable that depends on the generated sample path. The first performance measure is $\mathrm{E}\overline{Y}_1(t)$.

PM2. For some fixed $n \geqslant 1$ let

$$\overline{W}_1(n) = \frac{1}{n}\sum_{i=1}^{n} W_1(i)$$

be the mean waiting time in queue of the first n Class 1 jobs to arrive to the system. The second performance measure is $\mathrm{E}\overline{W}_1(n)$ for some fixed n.

PM3. Instead of expected values, we may wish to compute a tail probability such as $\Pr(\overline{W}_1(n) > 10)$.

PM4. The pth quantile of a random variable Z can be defined as $\inf\{z: \Pr(Z \leqslant z) \geqslant p\}$. We may wish to compute the pth quantile of $\overline{Y}_1(t)$ for some fixed p.

These are all examples of *finite-horizon* performance measures. An alternative term that is often used is *terminating-simulation* performance measures. The distinguishing feature of such measures is that they are related to the distribution of a sample path of finite, but possibly random, length. For example, PM1 requires one to simulate for a fixed length of simulated time, while PM2 requires one to simulate for an amount of simulated time that will vary from sample path to sample path. Both are finite-horizon performance measures. Both PM1 and PM2 are expected values of a particular random variable. One might also be interested in the distributions of these random variables beyond the mean. Both PM3 and PM4 provide such information.

Chapter 8 discusses how to estimate performance measures. Perhaps the most straightforward method is the *replication method*, which works as follows. We have previously described a "recipe" for generating a sample path over a finite interval. If this recipe is repeated n independent times, then one obtains n i.i.d. observations of the sample path. The replication method combines these i.i.d. observations to estimate the performance measure of interest. For example, if $\overline{Y}_1(t; i)$ is the value of $\overline{Y}_1(t)$ on the ith replication, then one can estimate $\mathrm{E}\overline{Y}_1(t)$ via the sample average

$$\frac{1}{n}\sum_{i=1}^{n} \overline{Y}_1(t; i). \tag{1}$$

It is important to distinguish between *within-replication* data and *across-replication* data in the replication method. Each sample path that is simulated consists of within-replication data. These data are often dependent. For example, the number of Class 1 jobs in queue at time 1 and at time 2 are dependent, so that $\overline{Y}_1(t; i)$ is a continuous-time average of a series of dependent observations for any fixed i. On the other hand, across-replication data consists of values from different replications. For example, if, as above, the replications are simulated independently, then $\overline{Y}_1(t; i)$ is independent of $\overline{Y}_1(t; j)$ for any

$i \neq j$, so that the (sample) average (1) is an average of independent observations.

The performance measures PM1–PM4 are all finite-horizon performance measures. One might also be interested in *infinite-horizon* performance measures. These are performance measures that depend on the distribution of a sample path that, in principle, is simulated over the entire interval $[0, \infty)$. One such performance measure is expected discounted performance.

PM5. For our example this might take the form

$$\mathrm{E} \int_0^\infty \mathrm{e}^{-\delta s} Y_1(s)\, \mathrm{d}s,$$

where $\delta > 0$ is a discount factor.

Although there is no need to specify a time horizon t for this performance measure, one must specify the discount factor δ. Estimating PM5 is not straightforward, since it depends on an infinite time horizon. Nevertheless, methods exist for estimating it.

All of the performance measures mentioned so far depend on the initial condition of the simulation. In other words, PM1–PM5 typically depend on the state and clock readings at time 0. While such performance measures are appropriate for many applications, there are some settings where they are not appropriate. For example, in some communication network applications, even a few seconds is a very long time in terms of network evolution. On such time scales, discounting may seem inappropriate, and the initial condition often plays essentially no role. In such settings, one may wish to turn to *steady-state* performance measures.

PM6. Suppose that

$$\overline{Y}_1(t) = \frac{1}{t} \int_0^t Y_1(s)\, \mathrm{d}s \to \alpha$$

as $t \to \infty$ almost surely, where α is a deterministic constant that is the same regardless of the initial condition of the simulation.

The constant α can be interpreted as the long-run average number of Class 1 jobs in the system. The *steady-state estimation problem* is the problem of estimating α. As with PM5, estimation of α poses several challenges to the simulation user. Methods have been developed to address those challenges, and are discussed in Chapters 13, 14, 15 and 16.

The term "steady-state" has been somewhat abused in this definition. The steady-state simulation problem as defined above would perhaps be better named the "time-average estimation problem", since the problem as stated involves estimating the limiting value of a time average. However, this definition is more easily stated than a more traditional definition involving steady-state distributions of stochastic processes. Furthermore, under certain conditions, limits of time averages like α above and the expected values of steady-state

distributions coincide. For our example those conditions are roughly as follows: The stations are capable of keeping up with the incoming work, the input processes are stationary in the sense that they do not change with time, and there are no peculiarities of the input processes that lead to periodic behavior. Under these conditions it is reasonable to expect that the distribution of the number of Class 1 jobs in the system $Y_1(t)$ at time t will depend less and less on t as t increases. Notice that the system state itself will continue to evolve in a random fashion, but the *distribution* of the system state will tend to settle down. Our definition of the steady-state estimation problem avoids the need to make some of these assumptions, which helps to explain our preference for it. See Wolff (1989) for more on this issue.

One of the key challenges in the estimation of any performance measure is error estimation. The term *error* refers to the difference between a deterministic quantity and an estimate of that quantity constructed through simulation. In simulation applications error estimation usually takes the form of a confidence interval for a performance measure. Consider PM1, for example, where we wish to compute the deterministic quantity $\mathrm{E}\overline{Y}_1(t)$ for some fixed t. Assuming that $\overline{Y}_1(t)$ has a finite variance, the central limit theorem applies, and ensures that the sample average (1) is approximately normally distributed for large n. This limit theorem then allows one to report a confidence interval for $\mathrm{E}\overline{Y}_1(t)$ and hence to get a sense of the error in the estimator. This form of error is often called *estimation error* to distinguish it from *modeling error* as discussed below. See Chapters 2, 8 and 9, and Part V for further details on evaluating and controlling error.

A common misconception is to confuse error with *risk*. Continuing with PM1, suppose that the decision maker is worried about excessive queue lengths in the period $[0, t]$. The summary performance measure $\mathrm{E}\overline{Y}_1(t)$ gives some idea of the magnitude one might expect of $\overline{Y}_1(t)$, and so it offers some information on the potential risk involved. A 95% confidence interval for $\mathrm{E}\overline{Y}_1(t)$ might be reported as $(Z - H, Z + H)$ or $Z \pm H$, where Z is the midpoint and H is the halfwidth. This confidence interval is sometimes misinterpreted to mean that 95% of the time the random variable $\overline{Y}_1(t)$ will lie in the interval $(Z - H, Z + H)$. To see why this must be false, notice that the confidence interval width can be made arbitrarily small by increasing the number of simulation replications. The confidence interval will still be a 95% confidence interval, but now will be of the form $Z' \pm H'$, where the interval $(Z' - H', Z' + H') \subset (Z - H, Z + H)$ with high probability. So the level of confidence 95% has everything to do with estimation error, and nothing to do with risk.

A measure of risk for this example may instead be related to the *distribution* of $\overline{Y}_1(t)$. For example, one might estimate the $\Pr(\overline{Y}_1(t) \in (y_\ell, y_u))$, the probability that the (random) average number of Class 1 jobs lies in the deterministic interval (y_ℓ, y_u). This probability can be estimated by simulation, and a confidence interval generated to give some idea of the accuracy of the

estimate of the probability. For further discussion of these and related points see Chapters 9 and 14.

We are finally in a position to conduct a simulation of our model. For our specific implementation the interarrival times of Class 1 jobs are exponentially distributed with mean 110 seconds, as are the interarrival times of Class 2 jobs. The processing times at Station 1 are deterministic and equal to 100 seconds, as are the processing times at Station 2. We start the system off empty, with arrival clocks sampled from their respective distributions. We estimate $\mathrm{E}\overline{Y}_1(t)$ and $\mathrm{E}\overline{Y}_2(t)$, where $\overline{Y}_2(t)$ is the average number of Class 2 jobs in the system, up to time $t = 500$ hours. We performed 20 replications, where each replication consisted of running the model for 500 hours.

The 95% confidence interval for $\mathrm{E}\overline{Y}_1(t)$ was 0.9 ± 0.3, so that the average number of Class 1 jobs in the system was around 1! This looks like absolutely wonderful performance. Unfortunately, the 95% confidence interval for $\mathrm{E}\overline{Y}_2(t)$ was 33 ± 4, so that Class 2 jobs fared rather poorly. On a hunch we also ran the model for 20 replications of $t = 2000$ hours each. The confidence interval for $\mathrm{E}\overline{Y}_1(t)$ remained around 1, but the confidence interval for $\mathrm{E}\overline{Y}_2(t)$ was 108 ± 12. The mean number of Class 2 jobs increased by approximately four times when we ran the simulation for four times as long. This suggests that the queue of Class 2 jobs is growing as the simulation proceeds. In other words, the system is unstable.

When such behavior is observed, one typically assumes that there is not enough service capacity to keep up with incoming work. But if Class 1 jobs were never routed to Station 2, then we would have two separate single-server queues where the mean interarrival time (110) is greater than the mean service time (100). Such queues are known to be stable. So there is, indeed, enough service capacity. It is just that our priority policy at Station 2 is not a good one.

To diagnose the trouble we explore further. A look at the utilizations of the stations shows that Station 1 is utilized approximately 65% of the time, while Station 2 is utilized 100% of the time. This suggests that Station 1 is being starved of work while Station 2 is overloaded. If one watches an animation of the system (a standard feature in most graphical simulation software), one immediately sees the trouble. When a Class 1 job arrives and Station 1 is busy, the job moves to Station 2 if it can be served immediately. Otherwise it waits in Queue 1. Then, when Station 1 completes its current job, there is often no new Class 1 job to work on. So Station 1 has to wait until the next Class 1 job arrives, and so it is often idle. Meanwhile, Station 2 continues to help out with Class 1 jobs, even when the queue of Class 2 jobs gets long.

There are several ways to modify the policy to improve performance. There is a great deal of theory (see, e.g., Harrison, 1996; Bell and Williams, 1999; Meyn, 2001, 2003) that suggests that a policy that is close to the best that one can do is to instead have Station 2 work on Class 1 jobs only when there are at least x Class 1 jobs in the queue, where x is a *decision variable* that needs to be chosen. If x is large, then Station 2 will only assist Station 1 when the backlog of Class 1 jobs is large. In this case it is highly unlikely that Station 1 will be

left with no jobs to process. Indeed, if we take $x = 5$ then we get confidence intervals for $\mathrm{E}\overline{Y}_1(t)$ and $\mathrm{E}\overline{Y}_2(t)$ given by 2.4 ± 0.2 and 8 ± 2, where t is again chosen to be 500 hours and we performed 20 replications. This is far more reasonable! In this setting, x is known as a safety stock. Notice that if x is too large, then we can end up with long waiting times for Class 1 jobs which is undesirable. On the other hand, if x is too small, then the system cannot keep up with the arrival of Class 2 jobs.

So an important question is how to choose the integer x. Suppose we adopt the position that we want to minimize $5\mathrm{E}\overline{Y}_1(t) + \mathrm{E}\overline{Y}_2(t)$. The multiplier 5 is chosen to reflect the importance of completing Class 1 jobs quickly. This is an example of a discrete-optimization problem where the objective function is estimated via simulation. Such problems are the subject of Chapters 20 and 21. Of course, we can only get *estimates* of the objective function, and so with a finite amount of simulation effort we can never be certain that the reported optimal value of x is indeed optimal. Chapter 17 discusses the question of how to report a solution as optimal with a given level of confidence. See Henderson et al. (2003) for more on the safety stock selection problem.

One needs to interpret confidence levels or, more generally, error estimates, with care. An error estimate is a statement about the statistical properties of the model that was implemented. As such it is a statement about *estimation error*. Recall that estimation error is the difference between the *true* value of a performance measure for the simulation model and the *estimate* of that value resulting from running the simulation model. We can typically make estimation error very small by running the simulation for a long time. It is important to be aware that a small estimation error is no guarantee that the simulation results closely mirror reality.

To see why, notice that the implemented model typically contains a number of simplifications of the underlying system that is being studied. (This underlying system could either be *extant*, i.e., currently existing, or *conceptual*, i.e., proposed but not yet existing. We do not distinguish between these two cases.) For example, in our network model we may have ignored a small variation in the interarrival distribution depending on the time of day, or there may be an operator that is required to set the processing in motion before a job can begin service that we have not modeled. Even if the model is a complete representation of reality, the estimation of parameters of distributions (e.g., the mean interarrival time of jobs) can introduce modeling error. The difference between the value of a performance measure for the *actual* system (whether extant or conceptual) and the value of the same performance measure for the corresponding simulation model is known as *modeling error*. One can simultaneously have small estimation error yet large modeling error.

The sum of these two sources of error gives the difference between true system performance and the estimate of performance based on the simulation model. One can often control estimation error by increasing simulation lengths. It is typically much more difficult to get a sense of modeling error. One way is to perform sensitivity analyses of the model, but there are also

other methods. The modeling error/estimation error issue is further explored in Chapters 9 and 14.

2.2 A stochastic activity network problem

Consider a set of nodes $\mathcal{N}$ and arcs $\mathcal{A}$ such that $(\mathcal{N}, \mathcal{A})$ defines a directed, acyclic network with a single source node $a \in \mathcal{N}$ and sink node $z \in \mathcal{N}$. Figure 2 is an illustration. Associated with each arc $i \in \mathcal{A}$ is a random variable X_i representing the "length" or "duration" of the arc. Here the arcs represent tasks to be completed in a project, and the nodes represent milestones along the way. Let $\mathcal{P}$ be the set of all distinct paths (sequences of arcs) from a to z. In the illustration $\mathcal{P} = \{(1, 4, 6), (1, 3, 5, 6), (2, 5, 6)\}$. There are at least two problems that can be associated with the network $(\mathcal{N}, \mathcal{A})$:

Stochastic longest route problem: If all inbound tasks to each node must complete before any outbound tasks can begin, then the time to complete the project is

$$Y = \max_{P \in \mathcal{P}} |P|, \tag{2}$$

where

$$|P| = \sum_{i \in P} X_i$$

is the duration of the path P, so that Y is simply the duration of the longest path. The random variables

$$C_j = I\big(|P_j| = Y\big), \quad P_j \in \mathcal{P}, \tag{3}$$

indicate which path (or paths) are the longest (most critical). Here, $I(\cdot)$ is the indicator function that is 1 if its argument is true and 0 otherwise.

Stochastic shortest route problem: If the paths $\mathcal{P}$ represent a collection of alternative approaches to the project, then the shortest route is of interest. When all approaches will be attempted simultaneously,

$$Y = \min_{P \in \mathcal{P}} |P|$$

is the time to complete the project. If only a single alternative will be

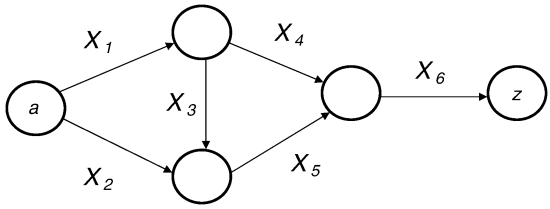

Fig. 2. Stochastic activity network.

chosen, then

$$C_j = I\big(|P_j| = Y\big)$$

is an indicator of whether path P_j is the shortest (optimal) path.

In this section we consider the stochastic longest route problem and focus on the completion time Y. The random variable Y is easier to simulate than the dynamic stochastic process of Section 2.1, since the following algorithm will suffice:

(1) Generate a random sample X_i for all $i \in \mathcal{A}$.
(2) Set $Y = \max_{P\in\mathcal{P}} |P|$.

Notice that the concept of time does not play a central role here as it did in Section 2.1; in fact, the simulation is merely an algorithm for realizing the random variable Y. Simulations of this type are sometimes called *Monte Carlo simulations*, although the term is also used for any simulation that involves random sampling. Placing this problem within the framework of Section 2.1, the stochastic longest route problem is a terminating simulation, and Y represents finite-horizon performance. Therefore, the performance measures PM2–PM4 are still relevant:

PM2. $\mathrm{E}Y$ is the expected value of the time to complete the project.
PM3. $\Pr\{Y > t\}$ is the probability that the project duration exceeds some threshold.
PM4. For given probability p, $y_p = \inf\{y\colon \Pr(Y \leqslant y) \geqslant p\}$ is the pth quantile of project completion, a completion date that can be achieved with probability p.

Let $\mathbf{X} = (X_1, X_2, \ldots, X_k)$ be the random vector of all task durations, with joint cumulative distribution function $F_{\mathbf{X}}$. Then we can write

$$F_Y(t) \equiv \Pr\{Y \leqslant t\} = \int_{\mathbf{x}:|P|\leqslant t, P\in\mathcal{P}} \mathrm{d}F_{\mathbf{X}}(\mathbf{x}). \tag{4}$$

If we could compute F_Y then, in theory, we could also compute $\mathrm{E}Y$ and y_p. However, even in simple networks this can be a difficult calculation, and it is essentially impossible if the network is large and the joint distribution of $\mathbf{X}$ is complex. As a result, simulation becomes competitive because PM2–PM4 can all be estimated by the replication method. For instance, if we repeat the simulation algorithm n times to produce i.i.d. samples of project completion time $Y_1, Y_2, \ldots, Y_n$, then the natural estimator for $\mathrm{E}Y$ is

$$\frac{1}{n}\sum_{i=1}^{n} Y_i, \tag{5}$$

the natural estimator of $\Pr\{Y > t\}$ is

$$\frac{1}{n}\sum_{i=1}^{n} I(Y_i > t) \tag{6}$$

and the natural estimator of y_p is

$$Y_{(\lceil np \rceil)}, \tag{7}$$

where $Y_{(1)} \leqslant Y_{(2)} \leqslant \cdots \leqslant Y_{(n)}$ are the order statistics (sorted values) of the sample.

If the network is large and has many paths, or t and p are extreme, then the estimation error associated with (5)–(7) may be unacceptably large unless n is very big. Part IV describes techniques for reducing the estimation error of the natural estimators without a corresponding increase in computational effort (larger n) or analyst effort (since every minute spent figuring out a more efficient estimator could be spent simulating).

Consider $\mathrm{E}Y$. Because (5) is an unbiased estimator, its mean squared error is

$$\begin{aligned} \mathrm{MSE}\left(\frac{1}{n}\sum_{i=1}^{n} Y_i\right) &= \mathrm{Bias}^2\left(\frac{1}{n}\sum_{i=1}^{n} Y_i\right) + \mathrm{Var}\left(\frac{1}{n}\sum_{i=1}^{n} Y_i\right) \\ &= \frac{\mathrm{Var}(Y)}{n}. \end{aligned} \tag{8}$$

Reducing this MSE is the goal of Chapters 10–12, and it is important to consider MSE because some efficiency improvement techniques reduce variance at the cost of introducing some bias. Efficiency improvement often requires that the analyst exploit knowledge beyond the minimum required to build the simulation. For instance, in the network of Figure 2 there is no need to simulate X_6 if the goal is to estimate $\mathrm{E}Y$: The $\mathrm{E}X_6$ can simply be added to the time to reach the penultimate node and a source of variance in the estimator disappears. Rarely is efficiency improvement this easy. Notice, however, that the expected duration of each path $P_j \in \mathcal{P}$ is computable since

$$\mathrm{E}\left(\sum_{i \in P_j} X_i\right) = \sum_{i \in P_j} \mathrm{E}X_i.$$

The actual durations of the paths with the longest expected durations will be correlated – perhaps strongly correlated if there are a few dominant paths – with the project duration Y. A more general efficiency improvement method, called control variates (see Chapter 10), can exploit the correlation between a random variable whose expected value is known and a random variable whose expected value we want to estimate to reduce estimator variance.

In the classical stochastic longest route problem $X_1, X_2, \ldots, X_k$ are independent, but this certainly need not be the case in practice. If the duration of several tasks are affected by weather (as might occur in a construction project), then particularly bad or good weather might influence several tasks, making their durations dependent. One reason that independence is often assumed is that characterizing and simulating a joint distribution, especially when the marginal distributions are from distinct families, is difficult. For instance, specifying the marginal distributions and correlations among $X_1, X_2, \ldots, X_k$ does

not, in general, uniquely define their joint distribution. Chapter 5 takes up this problem. Notice that dependence across the activity durations X_i does not interfere with the application of control variates described above, since only the expected path duration is required. On the other hand, the joint distribution could be critical to the definition of a path-based control variate for estimating $\Pr\{Y > t\}$.

Up to this point we have treated the network $(\mathcal{N}, \mathcal{A})$ as fixed. But it may be that, by spending some money, the duration of the project can be shortened. As a simple example, suppose that all of the task durations are exponentially distributed with nominal means $\mu_1, \mu_2, \ldots, \mu_k$, respectively, and that μ_i can be reduced to m_i at a cost of $\$c_i$ per unit reduction. If a total budget of $\$b$ is available, how should it be spent? We now think of the project duration as a random function of the mean durations m_i, say $Y(m_1, m_2, \ldots, m_k)$. (It is random because we have only specified the means of the task durations. The actual task durations are still random.) One way to formulate the decision problem is

$$\begin{aligned} &\min \mathrm{E}\big[Y(m_1, m_2, \ldots, m_k)\big] \\ &\text{subject to:} \\ &\sum_{i=1}^{k} c_i(\mu_i - m_i) \leqslant b \\ &0 \leqslant m_i \leqslant \mu_i,\ i = 1, 2, \ldots, k. \end{aligned} \tag{9}$$

This is an example of optimization via simulation, covered in Part VI, and specifically continuous-decision-variable optimization. Chapter 18 describes methods for selecting and fitting a tractable metamodel that represents, say, $\mathrm{E}[Y(m_1, m_2, \ldots, m_k)]$ as a function of the decision variables. Chapter 19 reviews techniques for estimating gradient search directions for continuous-decision-variable optimization. In both approaches the need to handle high-dimensional (large k) problems and sampling variability is central.

3 Organization of the Handbook

The remainder of this volume is organized into parts that contain several chapters each. Part I, including this chapter and the next, provides fundamental background that will be helpful when reading the more specialized chapters that follow. Part II covers random-number generation, random-variate generation, and issues related to the implementation of each. The two chapters in Part III lay the foundation for evaluating system performance via simulation from both a frequentist and Bayesian statistical perspective. Using the material in Part III as the foundation, Part IV describes methods for improving the computational and statistical efficiency of performance estimation, while Part V addresses the need to evaluate and control whatever estimation error

remains. Finally, Part VI tackles the goal of optimizing the performance of a simulated system with respect to controllable decision variables.

Acknowledgements

The work of the first author was supported by National Science Foundation Grants DMI-0224884, DMI-0230528 and DMI-0400287. The work of the second author was supported by National Science Foundation Grants DMI-0217690 and DMI-0140385.

References

Banks, J. (Ed.) (1998). *Handbook of Simulation*. Wiley, New York.

Banks, J., Carson II, J.S., Nelson, B.L., Nicol, D.M. (2004). *Discrete-Event System Simulation*, 4th edition. Prentice Hall, Upper Saddle River, NJ.

Bell, S., Williams, R. (1999). Dynamic scheduling of a system with two parallel servers: Asymptotic optimality of a continuous review threshold policy in heavy traffic. In: *Proceedings of the 38th Conference on Decision and Control*. Phoenix, AZ, pp. 1743–1748.

Fishman, G.S. (2001). *Discrete-Event Simulation: Modeling, Programming and Analysis*. *Springer Series in Operations Research*. Springer-Verlag, New York.

Fishwick, P.A. (1995). *Simulation Model Design and Execution: Building Digital Worlds*. Prentice Hall, Englewood Cliffs, NJ.

Fujimoto, R.M. (1999). *Parallel and Distributed Simulation Systems*. Wiley, New York.

Glasserman, P. (2004). *Monte Carlo Methods in Financial Engineering*. Springer-Verlag, New York.

Glynn, P.W. (1989). A GSMP formalism for discrete event systems. *Proceedings of the IEEE* 77, 14–23.

Haas, P.J. (1999). On simulation output analysis for generalized semi-Markov processes. *Communications in Statistics: Stochastic Models* 15, 53–80.

Harrison, J.M. (1996). The BIGSTEP approach to flow management in stochastic processing networks. In: Kelly, F.P., Zachary, S., Ziedins, I. (Eds.), *Stochastic Networks Theory and Applications*. Clarendon Press, Oxford, UK, pp. 57–89.

Henderson, S.G., Meyn, S.P., Tadić, V. (2003). Performance evaluation and policy selection in multiclass networks. *Discrete Event Dynamic Systems* 13, 149–189. Special issue on learning and optimization methods.

Jain, R.K. (1991). *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. Wiley, New York.

Law, A.M., Kelton, W.D. (2000). *Simulation Modeling and Analysis*, 3rd edition. McGraw-Hill, New York.

Meyn, S.P. (2001). Sequencing and routing in multiclass queueing networks. Part I: Feedback regulation. *SIAM Journal on Control and Optimization* 40 (3), 741–776.

Meyn, S.P. (2003). Sequencing and routing in multiclass queueing networks. Part II: Workload relaxations. *SIAM Journal on Control and Optimization* 42 (1), 178–217. Presented at the 2000 IEEE International Symposium on Information Theory, Sorrento, Italy, June 25–June 30.

Nelson, B.L. (2002). *Stochastic Modeling: Analysis & Simulation*. Dover Publications, Mineola, NY.

Shedler, G.S. (1993). *Regenerative Stochastic Simulation*. Academic Press, Boston, MA.

Sterman, J.D. (2000). *Business Dynamics*. McGraw-Hill, New York.

Wolff, R.W. (1989). *Stochastic Modeling and the Theory of Queues*. Prentice Hall, Englewood Cliffs, NJ.