Proceedings of the 1991 Winter Simulation Conference
Barry L. Nelson, W. David Kelton, Gordon M. Clark (eds.)

INTRODUCTION TO SIMULATION

Arne Thesen

Department of Industrial Engineering
University of Wisconsin-Madison
Madison, Wisconsin, 53706

Laurel E. Travis

Department of Finance and Management Science
University of Alberta
Edmonton, Alberta, Canada T6G 2R6

ABSTRACT

We give an overview of simulation modeling and analysis from the perspective of prospective users wanting to use simulation as a decision aid. Important considerations in building simulation models and analyzing their outputs are discussed. A few examples of simulation software are given. At the end of this tutorial, you should have a general understanding of simulation and its applicability to your situation.

1 INTRODUCTION

Simulation is simply the use of a computer model to "mimic" the behavior of a complicated system and thereby gain insight into the performance of that system under a variety of circumstances. Simulations are often used to determine how some aspect of a system should be set up or operated.

For example, we may want to understand how a proposed new runway at an airport will affect the percent of planes that must circle for a given length of time. To arrive at this understanding, we would first build a computer model representing the arrival and handling of planes. This model would use random variables to replicate variability in quantities such as the time between arrivals, the time required for an individual plane to land, and changes in the weather. Then we would run the model (i.e., operate the simulated airport), accumulating data on the delays experienced by individual simulated planes. This data would then be used as a basis for a cost-benefit analysis of the proposed runway.

As a second example, we may want to understand how the configuration of resources in a hospital will affect the time it takes for a patient to receive a variety of services. In this case, we would build a computer model to mimic the arrival of patients and the processing of the differing needs of the various patients. Processes such as admitting patients, taking X-rays, and drawing blood would be modeled as time delays. The computer model would keep track of the time patients spent waiting for various resources. We could then run the model and observe how well the resources are utilized, and how long patients must wait for service. Resources (such as additional X-ray machines) could be added or removed, and the resulting performance differences could be analyzed.

Or, we may be interested in the layout of a production line. Our model could help us understand how the routing of parts, the material-handling system, and the number and speed of machines of various types affect throughput and the quantity of work-in-progress inventory.

In these examples, as in most simulations, the computer program simply describes how the system operates and collects data on the performance of the system. It is the job of the analysts to propose scenarios to be tested and to draw conclusions from the simulation results.

1.1 What is Discrete Event Simulation?

We frequently hear of simulation being used for tasks such as driver training, rocket flight analysis, and weather prediction. These simulations describe how a system changes *continuously* over time in response to continuous controls (such as the turning of the car's steering wheel) that may vary smoothly through time. In contrast, *discrete event simulation* (the topic of this tutorial) describes systems that are assumed to change instantaneously in response to certain sudden or *discrete* events or occurrences. For example, if we were doing a capacity planning study involving a grain elevator, we might simulate how the quantity of grain stored changes over the course of a year. For this purpose, we would probably model the arrival of a truckload of grain as a discrete event. In other words we would ignore the fact that the quantity of grain stored changes slowly while grain is being poured into the storage facility and assume instead that the grain level jumps up to the new value instantaneously at a specific point in time. This assumption would be appropriate since we are modeling the system on a day-by-day, truckload-by-truckload basis instead of a second-by-second, grain-by-grain basis.

When we choose to model a real world system using discrete event simulation, we give up the ability to capture a degree of detail that can only be described as smooth continuous change. In return, we get a simplicity that allows us to capture the important features of many systems that are too complex to capture with continuous simulation.

1.2 Drawbacks and Pitfalls of Simulation

Simulation analysis is not without drawbacks. First, the quality of the analysis depends on the quality of the model. Second, it is often difficult to determine the extent to which an observation made during a simulation run is due to a significant underlying relationship in the system being modeled or due to the built-in randomness of the run; simulation results are hard to interpret. Finally, simulation is usually a time-consuming and expensive process, and an adequate analysis may not be feasible within the time available; analytic methods may be better for "quick and dirty" estimates. These and other pitfalls are listed in Figure 1.

1. Failure to state a clear objective
2. Failure to frame an answerable question
3. Using simulation when a simpler approach suffices
4. Inappropriate level of complexity
5. Bad assumptions in model
6. Misinterpreting simulation outputs

Figure 1: Common Pitfalls in Simulation

For simulation to be effective, it must be focused on a well defined problem (otherwise we do not know what elements of the system to include in the model and what information to collect). Using simulation before a specific problem is articulated may lead to a large number of unfocused simulation runs that use inappropriately designed models, and produce little or no information of value. This is perhaps the most common pitfall of simulation analysis. To avoid this all too common waste of time and money, a simulation project should not be undertaken unless there is a clearly defined question to be answered or decision to be made. This question or decision should be used to guide the development and analysis of the simulation model.

It is also important to use a model with an appropriate level of detail. Long term planning projects often evolve through a series of stages with the level of detail increasing at each stage. For example, questions about overall plant capacity are frequently asked early in the project when few details about the design are available, and fairly rough answers may suffice. In this case a simple model is appropriate. On the other hand, questions about the efficiency of different scheduling rules in an automated manufacturing line can only be answered when the detailed design of the system is finalized. In this case a detailed model is required and a fairly sophisticated analysis of the simulation output is called for.

When beginning a simulation, it is often tempting to build a model describing all of the phenomena that are easily observed. For example, if we want to understand the effect of the reliability of a given machine on overall throughput in a plant, we may be tempted to describe in detail how the machine works. This may be inappropriate, as the level of detail and time resolution required to completely describe machine operation is different from that of describing the general pattern of machine failure. It is therefore a good idea to begin with the simplest possible model that provides the necessary information. Starting with such a rough model enables the modeler to describe some of the important relationships in the system without excessive detail. The insights gained from this model can then be used to aid in the effective development of a more detailed model.

2 ELEMENTS OF SIMULATION MODELING

Our ability to develop simulation models of a wide range of different phenomena is due to the fairly universal nature of the building blocks on which the models are based. In particular, the representation of dynamic behavior and the use of random variables are fundamental to all discrete event simulations. These two concepts are discussed in this section.

2.1 Modeling Elementary Random Processes

Our goal is to mimic real life phenomena in the computer. For example, if we are studying the effect of different repair policies for factory equipment, we need to generate intervals between machine breakdowns that represent the intervals observed in the factory. Instead of carrying out a detailed analysis of the state of each machine in the system so that we can predict the exact time of specific breakdowns, we use random variables to mimic the overall *pattern* of breakdowns regardless of cause. The time of any one simulated breakdown will be different from what we observe in real life but the long range pattern of breakdowns should be indistinguishable from the real life process.

Most simulation models use random variables this way to compensate for our lack of detailed knowledge of what is going to happen at any one instant in a real life process. Given a phenomenon that we intend to model with a random variable, we must select an appropriate

probability distribution. The computer will then be programmed to generate random variates (observations of random variables) from this distribution for use in the simulation. The selection of appropriate probability distributions is critical to the art of model building. If we draw a data set of random observations from the distribution we have selected for our model, we want that data set to be statistically indistinguishable from empirical observations of the phenomenon we are modeling.

We are frequently asked to simulate situations about which we have limited knowledge -- we cannot fit a distribution to the data when there is no data. For example, we may be asked to evaluate the effect of different scheduling policies in a not-yet-constructed production system. Many modelers, in this situation, would select an exponential distribution for the random variable representing service time simply because this is a distribution that is well known and easy to work with. This choice, however, might severely decrease the accuracy of the model since exponential distributions tend to over-estimate the variability of a process. While incorrect variability may seem like a minor oversight when the mean of the distribution is correct, it can in fact cause extremely misleading results. Systems with lower variability tend to run much more smoothly and have fewer bottlenecks than systems with higher variability, so using distributions with inappropriately high variability can lead to pessimistic models and wasteful recommendations. Thus we see that selecting appropriate distributions in the absence of good data requires a great deal of experience and judgement, or the gathering of additional information.

In sum, we use random variables to mimic real world events in the computer. The choice of probability distributions for these random variables involves collecting data on the real world processes and fitting distributions to this data. Since the choice of these distributions has a large impact on the validity of the model, it is well worth the modeler's time and effort to collect good data.

2.2 Describing Dynamic Behavior

Discrete event simulation models are run by tracing events over time, particularly those events that change the state of the system. Since we do not normally think of systems in terms of events and state changes, we usually use a simulation language or software package that allows us to represent the model more naturally. The computer then translates this to an event oriented approach (i.e., events and state changes) to actually run the model. One such "natural" approach, the transaction flow approach, will be described here.

Many simulations describe how *transactions* flow through a *block diagram.* For example, transactions may represent subassemblies (e.g., car transmissions), and the block diagram may show how these subassemblies flow from station to station in an assembly process. As a second example, transactions may represent customers (e.g., patients at a hospital) and the block diagram may show how these customers progress through multiple stages of service. Using a limited number of standardized building blocks to describe what happens to transactions, these languages are able to represent the behavior of a wide range of different systems.

The first block in a model often generates transactions. For example, transactions representing individual patients in a hospital might be generated at random time intervals. Once generated, a transaction immediately flows through the diagram until it hits some obstacle that causes it to be delayed. Eventually, conditions change and the delayed transaction is allowed to move again. Two important mechanisms that cause the flow of transactions to be impeded are explicit delays and blocking. Blocking usually occurs when a transaction wants to use a *resource* that is currently not available. For example, the transaction may want to receive the attention of a server that is busy serving somebody else. Since many transactions may be waiting for service, a running model may contain a large number of transactions simultaneously.

A feature of the transaction-flow approach is that resources (e.g., X-ray machines) are not always explicitly shown in the model. Instead, we show how the resource and the transaction interact. Accordingly, simulation languages provide blocks to request the use of a resource and blocks to release control of resources. A block diagram of a single server waiting line or *queueing* model is shown in Figure 2. (Here the "server" could be an X-ray machine.) Instead of explicitly showing the server, a request for service is represented by the SEIZE block and the release of the server is represented by the RELEASE block. This diagram corresponds to a seven line program in the popular simulation language called GPSS. When run, the program would simulate the arrival, service, and departure of patients. The analyst could use the program to collect data on the simulated system and use this data to make decisions about various system redesign possibilities.

Several different dialects of GPSS are available. However, the resulting model will in most cases be almost identical regardless of which version (such as GPSS/H or GPSS/PC) is used. Furthermore, since most major simulation languages use very similar approaches for modeling dynamic behavior, conceptually similar models will result if one of the other major languages (such as SIMSCRIPT, SIMAN, or SLAM) is used.

These simulation languages all use similar approaches for modeling dynamic behavior. (Information on these, and other simulation packages, can be found throughout this volume and in the references listed at the end of this paper.)

In this section we have focused on two basic ideas that are common to all discrete event simulation: the need to use randomness and the need to describe dynamic behavior. While these "underpinnings" are the same for all discrete event simulations, their implementation varies widely. In the next section we will describe some different types of computer packages used by simulators.

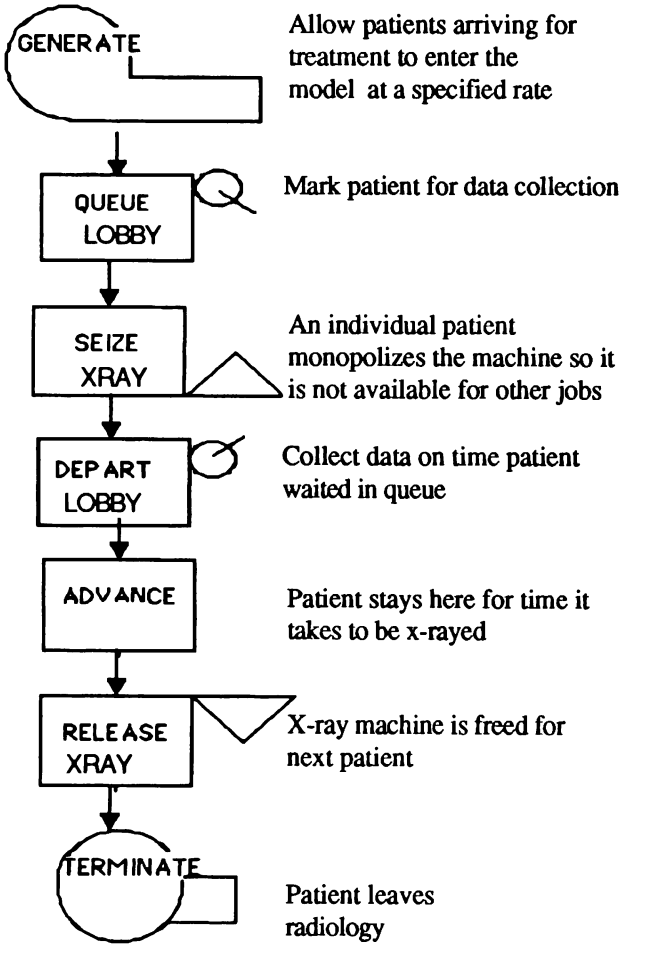

Figure 2: GPSS Block Diagram of a Single Server Queueing Model

3 SOME SOFTWARE PACKAGES

Since it can be exceptionally difficult to capture the detail of a situation or system on a computer, we try, whenever possible, to design models that draw upon previously developed models and programs. The more closely we follow a common model, the fewer choices we have to make. The further we deviate from commonly used models, however, the more difficult the modeling process becomes.

3.1 Template Packages

On one extreme, with special purpose simulators such as XCELL+, users are shown a model *template* giving model structure and default parameters. After changing a few parameter values, the model is ready to run. It is not necessary for the modeler to write a program describing the model structure, since this structure has already been built into the software. Although these template packages are convenient if the pre-programmed model accurately reflects the problem at hand, they are unable to describe many special circumstances.

As an illustration of a template based simulation system, consider the simple pallet loop system shown in Figure 3. A slab of processed meat is loaded onto a pallet at the loading station. The pallet transports the slab to the slicing station on an automated conveyor. At the slicing station, the slab is cut into individual slices and stacked on trays. The trays then move to the inspect station. If a problem exists, the entire stack is discarded. Good stacks are moved to the packing station. Here, stacks are packaged in a vacuum sealed plastic container. These packages leave the system and the empty pallet returns to the first work center.

This system could easily be simulated using a template system. If we used one such system, called TBS-II, we would use three different templates to describe model elements such as workstations, operations, and part flow. One of these, the workstations template, is shown at he top of Figure 4. Note that we specified limited buffer capacities simply by entering the size of the buffer in the appropriate field. Also note that unreliable machines are modeled simply by entering data about the failure and repair processes. Data is entered in the other templates in a similar manner.

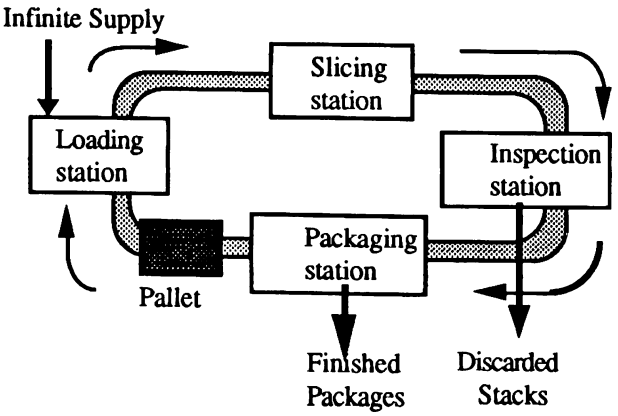

Figure 3: Simple Pallet-Loop System

Once all the parameters have been specified, the simulation is ready to run. Several different reports are produced at the end of the run. The workstation utilization report is shown at the bottom of Figure 4. With this information, bottlenecks can be spotted, and capacity decisions can be made.

Work Centers Input Template

	Work	Buffer		Machines		Reliability	
	Center Name	Size	Rule	count	Lot Size	MTTF	MRT
1	Load	5	fifo	1	2	8	0.25
2	Slice	5	fifo	2	1	9,999	0
3	Inspect	5	fifo	1	1	9,999	0
4	Pack	5	fifo	1	1	9,999	0

Work center utilization report (partial)

Center	Avg Buff Size	Avg Mach State				Mean transit time
		work	block	down	idle	
1 Load	0.51	0.50	0.00	0.03	0.47	1.6
2 Slice	0.01	1.40	0.00	0.00	0.60	1.4
3 Insp	0.45	0.72	0.00	0.00	0.28	1.1
4 Pack	2.84	1.00	0.00	0.00	0.00	3.8

Figure 4: Typical Input and Output from Luncheon Meats Example

This example is typical of template simulation in that the user needs to do no computer programming, but the system being simulated must "fit" the assumptions built into the template package. TBS, for example, is well suited for modeling the flow of parts between workstations, but of no use for modeling many other situations, such as inventory systems.

3.2 Simulation Languages and Animation

Users writing their simulations in a *simulation language* use model building blocks such as GENERATE, ADVANCE, TERMINATE, SEIZE, to specify the flow and logic of their model. While statements in a simulation language often correspond to activities in the system being modeled (ADVANCE, QUEUE) rather than to activities in the computer (multiply, divide), these languages have much of the structural flexibility of programming languages. Simulation languages therefore are appropriate when the modeler requires more flexibility than provided by template packages.

Many simulation languages can be used with graphical user interfaces (e.g., SLAMSYSTEM). These interfaces allow the user to write a simulation program by selecting blocks using pull-down menus and a mouse. Once a block is selected, the user is prompted to supply any accompanying parameters. The software then puts the blocks together in the order they were chosen and inserts any punctuation needed to create a simulation program with the correct syntax. These interfaces radically change what the model builder views on the screen, but they do not change the underlying logic or level of detail of the model; the modeler still requires the same detailed knowledge of how the model and the language work, and still retains the flexibility of a simulation language. While many beginning simulators find these graphical user interfaces an attractive way to avoid syntax and typographical errors, more advanced programmers often find it less cumbersome to type the program directly into a text editor or word processing package.

Software packages for animation of simulation models are also available for most of the common simulation languages. These packages allow the modeler to draw pictures representing the transactions and resources used in the model. When the simulation program is run, these pictures are used to create an animation that shows the events taking place in the model.

For example, the modeler might write a simulation program representing automatic guided vehicles moving pallets of materials on a factory floor. When the simulation program has been written, the modeler draws a static background depicting the layout of the factory floor and icons depicting the vehicles and pallets. When the simulation program is run, the animation package displays the static background on the computer screen and moves the vehicles and pallets through this background as dictated by the simulation program.

It should be noted that animation does not change the nature of the underlying simulation; it is simply a tool that aids in the debugging and presentation of a simulation model. But it can be a powerful tool. When used during program development, animation gives the programmer a simple overview of the internal workings of the program, allowing logical errors to be quickly pinpointed and debugging time to be dramatically reduced. When used with a finished model, animation helps the modeler present results to decision makers and convince them that the simulation adequately reflects the real-world system.

Animation, however, is not without pitfalls. Since the creation of a highly detailed animation can take days or weeks, the modeler should keep in mind that a rudimentary animation can be as useful as a visually elaborate one. The amount of time spent on purely aesthetic detail should usually be kept to a minimum.

Although the potential for wasted time is a serious pitfall of animation, even more alarming is the temptation for simulators using animation to perform

little statistical analysis on their finished models. An animated simulation can give the impression of a completed project; proposed modifications to the real-life system can be implemented in the simulation and the results can be viewed on the screen. A simulation project is not completed, however, until the model has been run many times and the results have been analyzed statistically. This fact can be difficult to remember when a polished animation is being viewed; intuition gathered directly from watching the animation can be wrong. A single run of the model may give the viewer the impression that a specific proposed system alteration has a positive effect when, in fact, random variation caused that particular run of the model to go smoothly. *Animation in no way replaces responsible, thorough analysis of simulation results.*

3.3 Other Software Tools

Occasionally a situation is so unique that it cannot be effectively modeled using a template package or simulation language. This might occur, for example, when complex material handling systems using fairly elaborate control schemes are simulated. If the control scheme used is quite complex, it may be cumbersome or even impossible to model it using the blocks provided by a simulation language. In these cases general purpose *programming languages* such as Fortran, Pascal, or C must be used. These languages were not designed for simulation, using them for simulation is often difficult.

Finally, there are certain types of discrete event simulation software that fall outside our classification scheme. For example, simulation models can be fully integrated with a factory's production scheduling system. When this integrated simulation is used, data giving the current status of the shop floor is input whenever the model is used. In this way, simulation can produce solutions that are specific to the problems of that particular day or even that particular instant.

For a comprehensive survey of simulation software see Law and Haider (1989), or since this information changes rapidly, consult the yearly proceedings of the Winter Simulation Conference.

4 INTERPRETING SIMULATION DATA

Regardless of the type of software used to perform the simulation, the use of simulation data for decision making should be approached with care. Since discrete event simulations are based on randomness, statistical analysis must be used to interpret simulation results.

Say, for example, that we are interested in the average time it takes a customer to get through a service facility (e.g., the radiology department of a small hospital). If we simulate this facility, we can easily record the length-of-stay for each individual customer during a run, and at the end of the run we can compute the *average* length-of-stay for customers in this run. If we run the program a second time, the computer will generate different observations of the random service and arrival times. Since the events that occur within the simulated system all depend on these times, the resulting average length-of-stay will be different from the first run. In fact, every time we run the simulation, we will in all likelihood observe a different average length-of-stay. We would like to draw conclusions and make decisions based on the average length-of-stay, but every time the model is run a different value results. We are observing values of a random variable whose distribution we do not know. In other words simulation follows the RIRO principle -- random input, random output. It is essential to keep this in mind when interpreting your results.

Since we deliberately introduce randomness at many different points in most simulation models, it is not surprising that the outputs from these simulations include randomness. Unfortunately, this randomness may cause the output from any one simulation to be of limited value. (Is the observed performance due to chance or is it due to some intrinsic property of the system?) We use statistical analysis to understand the effects of this randomness. Statistical tools for data analysis are used in a wide variety of different fields, and, generally speaking, tools that are useful in one area often work well in many other areas. However, as we summarize in Figure 5, several important differences exist between the way statistical tools are used in simulation and in other settings. As we will see in the next sections, a naive application of tools that work in other areas may give very misleading results when used to analyze simulation data.

	Other Contexts	Simulation
Collection	May introduce errors	Perfect
Outliers	Present	None
Randomness	Assumed	Under user control
Replications	Not always possible	Under user control
"Noise"	Often unknown origin	Fully explained
Scenarios	Often uncontrollable	Under user control
Underlying model	Unknown	Fully specified

Figure 5: How Simulation Experiments Differ From Other Statistical Experiments

4.1 Performance of a Single System

Say, for example, we want to determine whether the long run average length-of-stay in a system is greater than or less than 10. Perhaps we have run the model 20 times and collected this average during each run; for some runs the average was below 10, for some above. The overall average of these averages came out to, say, 10.3. Can we conclude that the overall long-run average (i.e., the true mean) is greater than 10, or would additional replications be likely to reverse this conclusion? Questions such as this are generally approached using statistical techniques such as confidence intervals and hypothesis testing. Simulation data, however, often present difficulties that make the computation of meaningful confidence intervals somewhat involved.

4.1.1 Autocorrelation

Most simple statistical data analysis techniques require that the data be observations of independent, identically distributed random variables. Simulation output data often does not satisfy this assumption. Consider, for example, the time it takes for a patient to wait in line at a busy X-ray machine. Two consecutive patients, due to their proximity, often wait for the same people ahead of them in the line. Consequently these two are likely to have similar waiting times and the corresponding data points are not independent (they are positively autocorrelated). The difference in the pattern of consecutive observations of independent and positively autocorrelated data is shown in Figure 6.

Observe how there is no obvious pattern in the independent data while large (and small) values tend to be clustered together in the positively autocorrelated data set.

This lack of independence between consecutive data points is an example of *autocorrelation*, a property of most simulation output data. Since autocorrelated data does not satisfy the assumption of independence, naive application of conventional statistical techniques will lead to misleading results. Changes often occur more slowly in positively autocorrelated data. If we ignore this, and naively compute the sample variance from a simulation data set, this estimate of variance is usually less than the true variance of the system. Confidence intervals based on small variance estimators are too narrow, and they may lead us to believe that our simulation results include much less error than they actually do. The actual percentage of time that a procedure for computing confidence intervals includes the true mean is referred to as the *coverage* of that procedure. The coverage of confidence intervals is often quite poor if the data is positively autocorrelated and that fact is ignored.

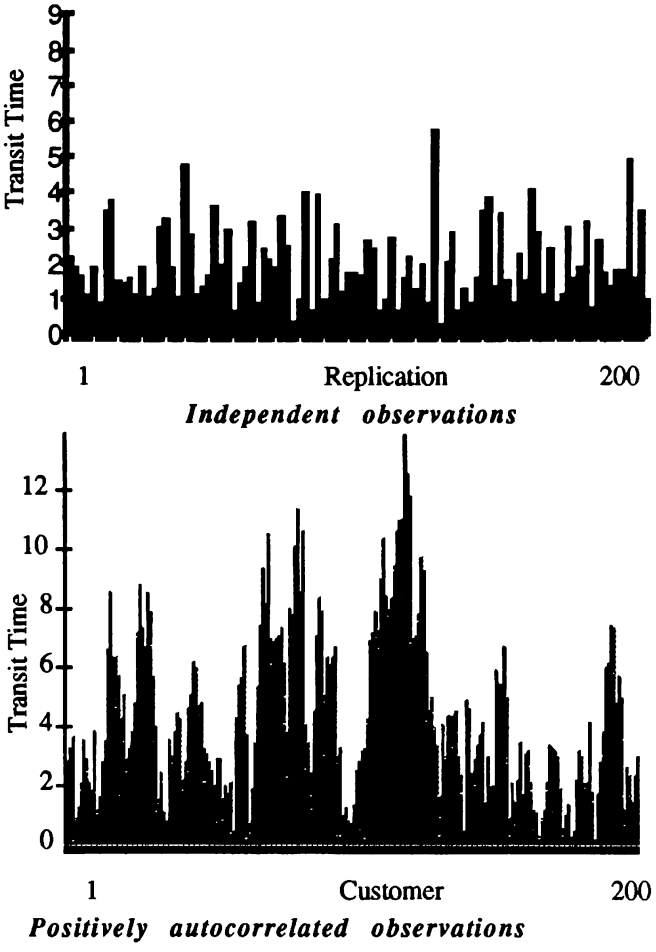

Figure 6: Two Types of Simulation Output Data

Several techniques have been developed for computing confidence intervals based on autocorrelated simulation data. Perhaps the simplest of these techniques, the method of *batch means*, is based on a simple idea. Assume, for example, that we have a data set composed of the time in the system for each of the first 12,000 customers in a simulation of a single server queueing system. We partition these 12,000 observations into twelve *batches* each containing one thousand consecutive observations. We then compute the average value of the observations in each batch, and base our confidence interval computation on this new twelve point data set. Our hope is these twelve averages (called *batch means*) are sufficiently independent to make the resulting confidence interval meaningful.

In Figure 7 we show that the coverage of batch means confidence intervals is substantially better than the coverage of the confidence intervals that result when the autocorrelation is simply ignored. To generate the data in this figure, a single server (M/M/1) queueing system with different levels of utilization was simulated.

Confidence intervals for the mean time in the system were computed, using a total of 12,000 departures to compute each interval. For the batch-means intervals, one thousand departures were used for each batch mean. Each point shows the fraction of 100 90% confidence intervals that contained the true mean.

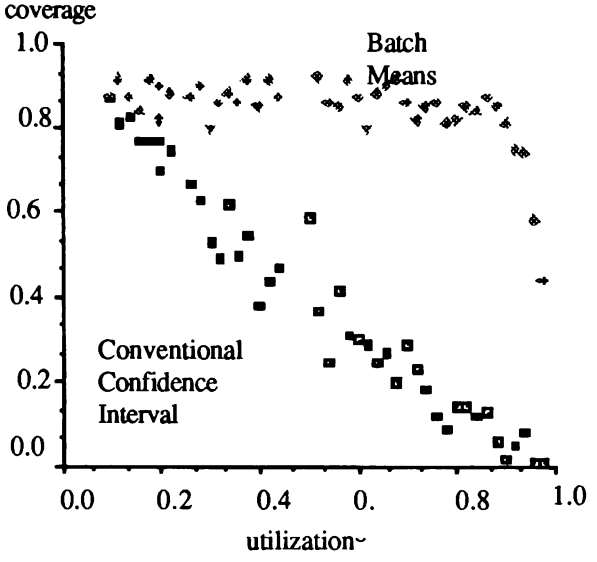

Figure 7: Coverage of Some 90% Confidence Intervals

We see that the batch means technique worked reasonably well for server utilization up to about 85% but that the coverage rapidly deteriorated as the utilization increased beyond this point. We also see that the coverage of conventional confidence intervals (computed without assuring that successive data points are independent) decreased at a constant rate regardless of server utilization. Thus we conclude that if the data set has significant autocorrelation (as in the case of time-in-system data for a queueing system), we should never compute confidence intervals straight from the original data. Batch means analysis can be used with reasonable accuracy unless the autocorrelation is very high (in our data, high autocorrelation was caused by high server utilization). More sophisticated techniques compensate more effectively for high autocorrelation.

4.1.2 The Initial Bias

Unfortunately, autocorrelation is not the only property of simulation data that makes analysis challenging. Another such property is known as the *initial bias*.

We often want to use simulation to evaluate the performance of fairly crowded and busy systems. For example, we may want to simulate an assembly system with fourteen different stations and several hundred parts in process at one time. It is often convenient to start such simulations with an empty model (i.e., no parts at any of the stations). This does not seem to be too unreasonable, since the model will quickly "fill up" and reach a representative state. Unfortunately, unless properly handled, this practice may cause serious distortions in the data that we collect. Since the first simulated parts flowing through the system encounter little or no congestion, these parts have shorter than average transit times. By including these values in our data set, we bias the average transit time downward.

To show the impact of initial bias, we computed five separate confidence intervals for the mean time-in-system for a simple queueing system. To compute the first confidence interval, we ran 240 independent replications of our simulation, each starting with an empty system, and for each replication we computed the average time in the system. These 240 independent averages were then used to estimate a confidence interval for the true mean of the time in the system.

We then repeated the entire process four more times, using a different number of replications each time (but each time using the same *total* number of observations). The resulting confidence intervals are shown in Figure 8. For comparison, we have also computed the true mean of our performance measure (mean time in system) using queueing theory and included it on the graph.

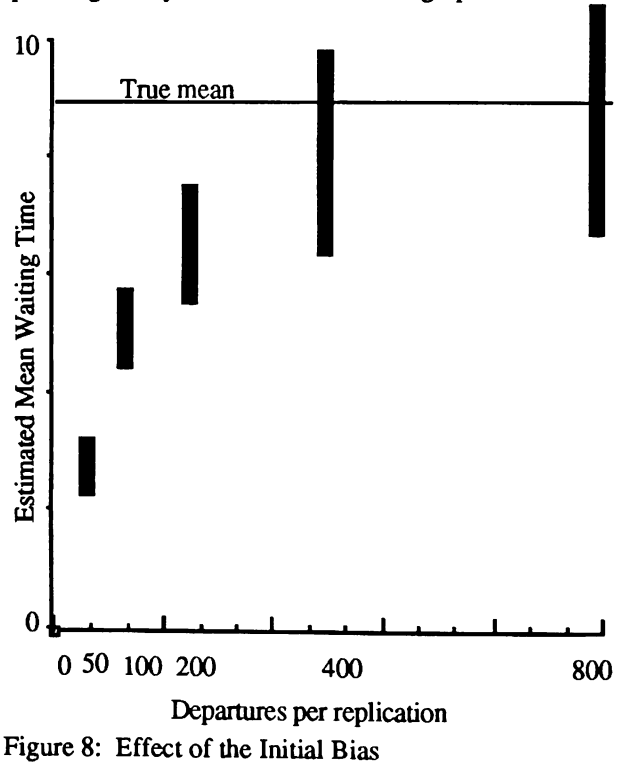

Figure 8: Effect of the Initial Bias

We see that the first confidence interval (based on the fifty departures per replication) gives information that is very precise (since the confidence interval is narrow), but also very incorrect (since the interval is very far away

from the true mean of 9.0 minutes). The analyst who made the mistake of believing this computation would have a result that seemed quite convincing but was, in fact, off by approximately a factor of three. This disturbing miscalculation is a consequence of initial bias. As the number of departures per replication increased, however, the confidence interval moved towards the true mean (since the effect of the initial bias was only present in the early observations of each run).

We see that failure to take into account the effect of the initial bias can lead to very serious estimation errors, especially when the simulation runs are short. The easiest way to reduce the effect of the initial bias is to run the model for a suitable length of time before beginning data collection. Other more sophisticated techniques for compensating for initial bias can be found elsewhere in this volume.

4.2 Comparing Designs

Simulations are often used to compare the performance of different potential solutions to a problem. This requires careful planning. Since the observed value of a performance measure for any single simulation run is an observation of a random variable, the observed difference between the performance of multiple systems will also be a random variable. Say we are interested in comparing system A to system B, and we run a computer simulation of each. If the simulated performance of system A is slightly better than that of system B, can we conclude that this difference is due to a genuine difference between the systems, or might it only be caused by the inherent randomness of our simulation runs?

As an example, recall the simple pallet loop system discussed earlier. A decision maker involved in the design of such a system might have two potential configurations in mind. These two configurations might differ, say, in the number of machines and buffer space at each work center. The decision maker may wish to know which of these systems is likely to produce a higher throughput in the long run.

Say we simulated the two potential setups and ran each simulation ten times collecting throughput data for each replication. We might for example get an average throughput of 1117 packages per hour for the first setup, and 1178 packages per hour for the second. Do we then conclude that, since 1178 > 1117, the throughput will, on average, be higher for the second setup and therefore this setup should be used? Shall we, on the other hand, say that the observed difference between the averages is too small to be conclusive given that only ten replications were made and additional runs might give the opposite answer?

We generally use hypothesis testing to answer questions such as these. In this example we would start from the hypothesis that the two setups result in the same average throughput, and then if our data provides strong enough evidence, we would reject this hypothesis, and conclude that the average throughput does indeed depend on which setup we use.

Sometimes our data does not provide enough evidence to reject the hypothesis we have stated, and the analysis is therefore inconclusive. This inconclusiveness is either due to the fact that there indeed is no difference in performance, or it could be due to the fact that the variance in the observed data was too high (i.e., the data was too "noisy" for us to extract useful information from a data set of this size). If the observed difference was due to the built-in randomness of the model, then we could either perform many more (perhaps expensive) simulation runs or we could use special techniques called *variance reduction* techniques to strategically collect a data set that is likely to be more conclusive.

In Figure 5 we pointed out that there are a number of important differences between simulation experiments and other statistical experiments. One of the most important of these is the fact that the random behavior observed in simulation experiments is completely under our control. There are many ways in which we can exploit this to increase the information gained in a simulation experiment. For example, we can use identical "random" streams of customers to evaluate two different management policies. Since the random factors are identical in the two simulations, the observed differences are more likely to be due to intrinsic differences between policies. Data collected using this technique is more likely to yield a conclusive statistical analysis.

Techniques such as the one described above are referred to as *variance reduction techniques*. Their use can significantly reduce the data requirement (and hence the cost) of many simulations. Variance reduction requires a thorough understanding of simulation and statistics; information on this topic can also be found in this volume.

We have seen that the RIRO principle (random input, random output) has implications critical to the interpretation of simulation results. Observed differences between individual runs of two simulated systems do not necessarily imply actual performance differences between the two models. Many replications of the simulation runs, careful data analysis, and perhaps variance reduction techniques, are required to get conclusive results.

In sum, since performance measures generated by simulations are random, any decisions made from simulation generated data must be based on statistical inference. However, simulation data often does not

satisfy the assumptions underlying the most common techniques for computing confidence intervals and performing hypothesis testing. Application of inappropriate statistical techniques may lead to misleading conclusions and hence expensive errors in policy. Consequently, a great deal of effort has been devoted to the development of valid, efficient techniques for extracting information from simulation data.

5 SUMMARY

We have attempted to provide an introduction to the uses of simulation, the underlying concepts, and the types of computer packages available to the analyst. While some of these tools require significant expertise and experience, others are quite accessible to the novice. Some guidelines for the beginning analyst are:

1) Define your objectives before simulating.
2) Use the correct level of detail -- begin with a simple model.
3) Select software that is appropriate for your problem, level of experience, and time frame.
4) Remember that simulation results are observations of random variables, and learn to interpret your results accordingly.

We have also pointed to a few of the many technical considerations involved in effective simulation. Finally some useful references are given in the following section.

6 FURTHER READING

For readers interested in a introductory overview of simulation methodology, a number of appropriate texts exist. Three of these are Banks and Carson (1990), Pegden, Sadowski, and Shannon (1990), and Thesen and Travis (1991). For more those interested in a more advanced methodology text, we recommend Bratley, Bennett and Schrage (1987), Johnson (1987), or Law and Kelton (1990). For a text that focuses on a specific simulation language, we recommend Banks, Carson, and Sy (1989), Markowits, Kiviat, and Villanueva (1987), Pegden, Sadowski, and Shannon (1990), Pritsker (1984), or Schriber (1990), depending on the language of interest.

ACKNOWLEDGEMENT

We would like to thank West Publishing Co. for allowing us to adapt material from our text *Simulation for Decision Making* for this tutorial.

REFERENCES

Banks, Jerry and John S. Carson II. 1990. *Discrete-Event System Simulation*, Second Edition. Englewood Cliffs, NJ: Prentice-Hall.

Banks, Jerry, John S. Carson II, and John Ngo Sy. 1989. *Getting Started With GPSS/H*, Wolverine Software Corporation, Annandale, VA.

Bratley, Paul, Bennett L. Fox, and Linus E. Schrage. 1987. *A Guide to Simulation*, Second Edition. New York: Springer-Verlag.

Johnson, M.E. 1987. *Multivariate Statistical Simulation*. New York: John Wiley.

Law, Averill M. and S. Wali Haider. 1989. Selecting Simulation Software for Manufacturing Applications: Practical Guidelines & Software Survey. *Industrial Engineering*, May 1989, 33-46.

Law, Averill M., and W. David Kelton. 1990. *Simulation Modeling and Analysis*, Second Edition. New York: McGraw-Hill.

Markowits, H.M., P.J. Kiviat, and R. Villanueva. 1987. *Simscript II.5 Programming Language*, CACI, Los Angeles, CA.

Pegden, C.D., R.P. Sadowski, and R.E. Shannon. 1990. *Introduction to Simulation Using SIMAN*, Systems Modeling Corporation, Sewickley, PA.

Pritsker, A.A.B. 1986. *Introduction to Simulation and SLAM II*, Third Edition. New York: Halsted Press.

Schriber, Thomas J. 1990. *An Introduction to Simulation Using GPSS/H*, Second Edition. New York: John Wiley.

Thesen, Arne and Laurel E. Travis. 1991. *Simulation for Decision Making*, West Publishing Co.

AUTHOR BIOGRAPHIES

ARNE THESEN is a professor and chair of the Department of Industrial Engineering at the University of Wisconsin-Madison. His research interests are in the areas of simulation modeling and scheduling of material handling systems.

LAUREL TRAVIS is an assistant professor in the Department of Finance and Management Science at the University of Alberta. Her research interests include applications of operations research to a variety of business and microeconomic theory settings.