Proceedings of the 2001 Winter Simulation Conference
B. A. Peters, J. S. Smith, D. J. Medeiros, and M. W. Rohrer, eds.

GRAPHICAL REPRESENTATION OF IPA ESTIMATION

Michael Freimer

School of Operations Research
and Industrial Engineering
Cornell University
Ithaca, NY 14853, U.S.A.

Lee Schruben

Department of Industrial Engineering
and Operations Research
University of California at Berkeley
4135 Etcheverry Hall
Berkeley, CA 94720-1777, U.S.A.

ABSTRACT

Infinitesimal Perturbation Analysis (IPA) estimators of the response gradient for a discrete event stochastic simulation are typically developed within the framework of Generalized semi-Markov processes (GSMPs). Unfortunately, while mathematically rigorous, GSMPs are not particularly useful for modeling real systems. In this paper we describe a procedure that allows IPA gradient estimation to be easily and automatically implemented in the more general and intuitive modeling context of Event Graphs. The intent is to make IPA gradient estimation more easily understood and more widely accessible. The pictorial nature of Event Graphs also provides insights into the basic IPA calculations and alternative descriptions of conditions under which the IPA estimator is known to be unbiased.

1 INTRODUCTION

A common issue that arises in discrete event simulation is how to find the gradient of a system performance measure with respect to some system parameter. For example, if θ is a parameter of the distribution of service times in a single-server queue, we may wish to find the derivative of the average customer waiting time with respect to θ. Such gradients may be required in a variety of contexts, such as stochastic optimization and output sensitivity analysis. For examples, see Fu and Hu (1997) and Glasserman (1991).

Infinitesimal Perturbation Analysis (IPA) is a technique for estimating the gradient of a system performance measure. Its primary advantage is that derivatives with respect to multiple parameters can be calculated from a single simulation run. By contrast, the method of finite differencing requires two simulation runs to calculate the derivative with respect to a single parameter θ; runs are made with parameter values θ and $(\theta+\Delta\theta)$ for some suitably small $\Delta\theta$. If $L(\omega,\theta)$ is a realization of the system performance measure with parameter value θ, the finite differencing gradient estimate is $\frac{1}{\Delta\theta}\left[L(\omega_1,\theta+\Delta\theta)-L(\omega_2,\theta)\right]$. In the calculation of an $n\times n$ Hessian matrix, IPA provides an $(n+1)$-fold computational savings.

The intuition behind IPA is demonstrated by the following example. Consider the derivative of the average waiting time $W(\theta)$ of customers in a single server queue with mean service time θ. Figure 1 shows a sample realization of this system. The area under the solid line is the overall waiting time of customers in the system. Jumps up represent arrivals; jumps down represent service completions.

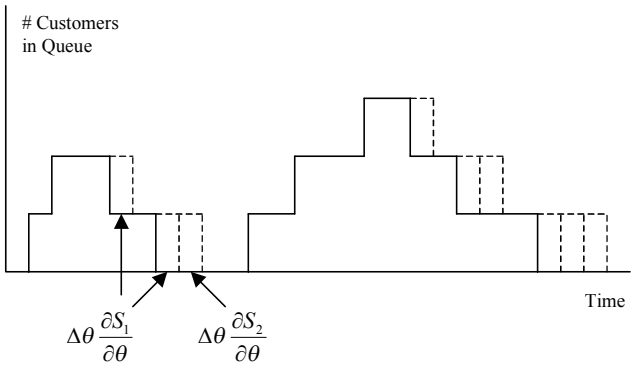

Figure 1: Sample Realization for a Single Server Queue

Let $S_i(\omega,\theta)$ be the service time of the i^{th} job. A small increase $\Delta\theta$ in θ will cause individual service times to increase by an amount $\Delta\theta\frac{\partial S_i(\omega,\theta)}{\partial\theta}+o(\Delta\theta)$. In Figure 1 this is represented by the width of one of the dashed rectangles. Since arrival times are unchanged, the waiting time of a particular job is only affected by the service times of jobs that come before it in the same busy period. For example, the waiting time of the third job in the busy period is lengthened by the increases in service time of the two jobs ahead of it. In general, the waiting time of a job

is increased by the sum of the increases in service time of the jobs ahead of it in the busy period.

If the change $\Delta\theta$ is small enough, the sequence of events in the simulation run will remain fixed, and no busy periods will merge. In this case we can calculate the effect on the overall waiting time of all jobs as follows. Suppose the simulation is run for M busy periods where the m^{th} busy period starts with job k_m. The change in overall waiting time due to $\Delta\theta$ is:

$$\sum_{m=1}^{M}\sum_{i=k_m+1}^{k_{m+1}-1}\sum_{j=k_m}^{i-1}\Delta\theta\,\frac{\partial S_j(\omega,\theta)}{\partial\theta} = \sum_{m=1}^{M}\sum_{j=k_m}^{k_{m+1}-1}(k_{m+1}-1-j)\Delta\theta\,\frac{\partial S_j(\omega,\theta)}{\partial\theta}.$$

In Figure 1 the total area under the dashed lines represents this quantity.

The primary rationale for IPA is that in the calculation of the derivative of $W(\theta)$ we let $\Delta\theta$ go to zero, so the order of events in the simulation run remains fixed. The sample path derivative can then be written as:

$$\begin{aligned}&\frac{dW(\theta,\omega)}{d\theta}\\ &=\lim_{\Delta\theta\to 0}\frac{1}{\Delta\theta}\left[\frac{1}{N}\sum_{m=1}^{M}\sum_{j=k_m}^{k_{m+1}-1}(k_{m+1}-1-j)\Delta\theta\,\frac{\partial S_j(\omega,\theta)}{\partial\theta}\right]\\ &=\frac{1}{N}\sum_{m=1}^{M}\sum_{j=k_m}^{k_{m+1}-1}(k_{m+1}-1-j)\frac{\partial S_j(\omega,\theta)}{\partial\theta}\end{aligned}$$

where N is the number of jobs served during the simulation run. So in order to compute $dW(\theta,\omega)/d\theta$, we do not choose a particular value for $\Delta\theta$; we only track sums of service time derivatives.

An immediate concern is how to interpret a "service time derivative," i.e. a derivative of a random variable. If we think of a random variable as generated by the method of inverse transformation, its realization is a function of the distribution parameter, θ, and of a random number, $U(\omega)$. For example, in the m/m/1 queue, where θ is the mean service time, we have $S(\omega,\theta)=F_S^{-1}(U(\omega),\theta)=-\theta\ln(1-U(\omega))$, and $\frac{\partial S(\omega,\theta)}{\partial\theta}=-\ln(1-U(\omega))=\frac{1}{\theta}S(\omega,\theta)$. In this case the service time derivative is a function of the service time itself. In other words, *we can compute the derivative directly from the observed service time.* As we shall see, this is an important requirement for implementing IPA. Suri (1987) and others have argued it also allows us to implement IPA not just for simulated systems, but for physical, real-time systems as well.

The IPA derivative calculated above is a sample path derivative. In other words, it is the derivative with respect to θ of the performance measure calculated for a particular realization of the sample path. However we may be interested in the derivative of the expected performance measure, $\partial E_\omega[L(\omega,\theta)]/\partial\theta$. A central question is whether the IPA sample path derivative is an unbiased estimator. In other words, under what conditions is it true that

$$E_\omega\left[\frac{\partial L(\omega,\theta)}{\partial\theta}\right]=\frac{\partial E_\omega[L(\omega,\theta)]}{\partial\theta}\,?$$

The conditions for unbiasedness and many other useful IPA results are often derived in the framework of generalized semi-Markov processes (GSMPs). See, for example, Fu and Hu (1997) and Glasserman (1991). GSMPs have a well-defined structure that facilitates formal proofs, and a variety of useful systems can be defined as GSMPs. However the GSMP structure is relatively awkward for model building. It may be difficult to see how a particular system can be modeled as a GSMP, and sometimes the generic framework has to be modified to fit a particular system; in these cases the IPA results must be tailored to fit as well.

In contrast, event graphs were introduced by Schruben (1983) specifically to facilitate model building for discrete-event systems. Event graphs are a general modeling paradigm that includes GSMPs as a subset. Schruben and Savage (1996) describe a direct translation from GSMPs to event graphs and provide an example of an event graph that cannot be described in the generic GSMP framework. Additional work concerning event graphs has been done by Som and Sargent (1989).

Event graphs, with their visual representation of the relationships between events, present a natural framework for implementing IPA. In Freimer and Schruben (2001) we compute an IPA derivative for event graph models and give conditions under which the derivative is an unbiased estimator. Our purpose in this paper is to describe how this computation can be easily implemented. In Section 2 we review the formal definition event graphs, and in Sections 3 and 4 we describe the implementation of IPA for event graphs and give an automatic procedure for implementation.

2 THE EVENT GRAPH FRAMEWORK

An event graph model consists of several components. The state of the system is described by a set of state variables. In the graph, a set of event vertices, V, represents the events, and a set of directed edges, D, represents the way in which events are scheduled. Associated with each

vertex $a \in V$ is a function $h_a(.)$ that describes the state changes caused by the event. The basic building block of the event graph is shown in Figure 2.

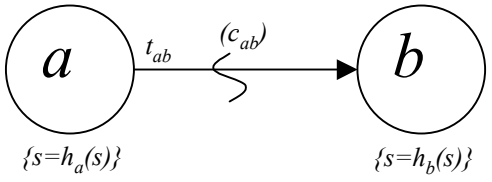

Figure 2: Event Graph Building Block

This construct indicates that "whenever event a occurs, the system state s changes to $h_a(s)$. Then, if condition c_{ab} is true, event b will be scheduled after a delay of t_{ab}." The event graph model can be interpreted by reading each component in a similar way. Appropriate labels are omitted if the time delay is zero or if the scheduling is unconditional. We use $F_{ab}(.)$ to refer to the distribution of t_{ab}; it is possible for b to be scheduled to occur without any simulated delay.

As an example, Figure 3 shows an event graph representation of a system with a single queue and two identical servers. Here the state variable Q is the number of waiting customers in the system, and S is the number of free servers. The random time between customer arrivals is denoted as t_a, and the random time of customer service as t_s. Figure 3 also shows a common feature of event graphs, a *Start* or *Run* vertex, which represents the first event to be executed. State variables are often initialized by this event.

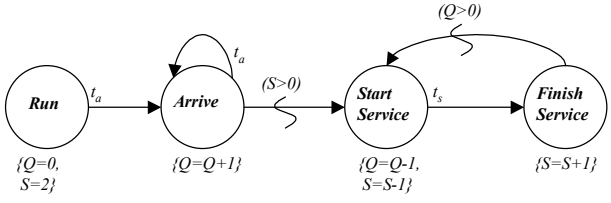

Figure 3: System with Two Servers, Single Queue

The evolution of an event graph model is similar in spirit to that of GSMPs, the modeling paradigm for which IPA results are usually derived. (For a formal definition of GSMPs, see Glasserman (1991).) As with the GSMP, the state variables remain constant except when an event occurs. (We sometimes say that an event is "executed.") Associated with the simulation run are a simulation clock and a future events list (FEL). The FEL is an appointment book that records information about events scheduled to take place after the current clock time. An element of the FEL is an ordered pair (a, C_a) where:

- $a \in V$ is the event type;
- C_a is the clock time at which the event is scheduled to take place.

The FEL may contain more than one event of the same type; for example, in the two-server queue shown above there will be two *Finish Service* events on the FEL when both servers are busy.

When the simulation run begins, the system clock is advanced to the time of the first event on the FEL. If this event is of type a, the state variables are updated according to function $h_a(.)$. The event may also schedule one or more additional events to take place in the future; these events are added to the FEL as follows. For each directed edge e_{ab} leading from event vertex a to another vertex b on the event graph, we evaluate an edge condition. If the condition is true, we add an instance of event b to the FEL with clock time: $C_b = C_a + t_{ab}$, where t_{ab} is drawn from distribution $F_{ab}(.)$. After the FEL has been updated, the current event a is removed from the list, and the system clock is advanced to the time of the next event on the list. The simulation executes this next event, updates the FEL, and continues in a similar fashion.

To simplify the results in the next section, we will assume that every event graph contains a *Run* vertex, and that the initial FEL contains only the *Run* event, scheduled at time 0. In practice this is a very mild assumption.

When it is possible for more than one event on the FEL to be scheduled for the same time, it may be important to specify the order in which the events are to be executed. If event a should be executed before event b, we say a has a higher *priority* than b. We assume the priority relationships among events satisfy a transitive property.

We now define some notation for use with event graphs:

- V: set of event types (vertices of the event graph);
- $X(a,n)$: n^{th} clock sample used to schedule an event of type $a \in V$;
- a_n: event type of the n^{th} event to occur during the simulation run;
- τ_n: epoch of the n^{th} event to occur ($\tau_0 \equiv 0$);
- s_n: n^{th} state visited by the process (s_0 is the initial state at time 0).

A *sample path* of length n is the sequence $\{(\tau_i, s_i), i = 0,\ldots,n; (a_i), i = 1,\ldots,n\}$. The clock sample $X(a,n)$ refers to the edge delay time $t_{\bullet,a}$ used to schedule the n^{th} instance of event a.

It is important to notice that for two or more events of type a, the order in which they are added to the FEL is not necessarily the order in which they are executed. In particular, the clock sample used to schedule the n^{th} execution of an event of type a is not necessarily the same sample used for the n^{th} addition of an event of type a to the FEL. Hence we define $I(a,n)$ to be the index of the clock sample used to schedule the n^{th} execution of an event of type a, an index into the series $\{X(a,1), X(a,2),\ldots\}$. If we define $N(a,n)$ to be the number of instance of an event of type a

up to and including the n^{th} event epoch, then $I(a_n,N(a_n,n))$ is the index of the clock sample used to schedule the n^{th} event, an index into the series $\{X(a_n,1), X(a_n,2), \ldots\}$. To ease the notation, we define $I(n) \equiv I(a_n,N(a_n, n))$.

We say that the i^{th} event is *triggered* or *scheduled* by a previous event numbered $\kappa(i)$ if the event graph contains a directed edge from $a_{\kappa(i)}$ to a_i that caused the i^{th} event to be added to the FEL at time $\tau_{\kappa(i)}$. Adapting Fu and Hu's notation (Fu and Hu 1997), the triggering event index set Ψ_i of a_i is defined recursively: $\Psi_i = \{\kappa(i)\} \cup \Psi_{\kappa(i)}$ if $i>1$, and $\Psi_i = \phi$ if $i=1$. The set Ψ_i contains the genealogy of event a_i. Using this notation we have:

$$\tau_n = \sum_{i \in \Psi_n} X(a_i, I(i)) \tag{1}$$

for $n \geq 1$.

To close this section, we describe an enrichment to the basic event graph setup: the ability to pass attribute values from one event into parameters of a scheduled event. The attribute values are determined when the originating event is executed and remain unchanged until the destination vertex assigns them to its parameters. These values are determined after the edge conditions are evaluated and stored on the FEL if the event is scheduled. Under this enrichment the FEL is now an ordered triple: (a, C_a, w_a) where $a \in V$ is the event type, C_a is the clock time at which the event is scheduled to take place, and w_a is the vector of attribute values.

A vector of parameters corresponding to the vector of passed attributes must be included in the description of the destination vertex. When executing an event, the assignment of parameters is done prior to any state changes. A pictorial representation of attributes and parameters is given in Figure 4. This construct indicates that "whenever event a occurs, the system state s changes to $h_a(s)$. Then, if condition (c_{ab}) is true, event $b(j)$ will be scheduled after a delay of t_{ab} with the parameter j equal to attribute value k." Typically k is a string of state variables, and j is a string of their future values.

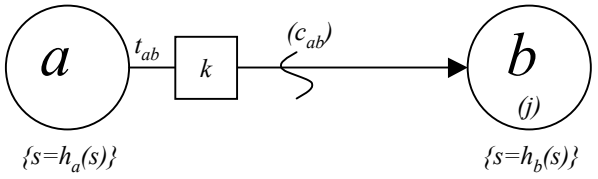

Figure 4: Event Graph Building Block with Parameters and Attributes

Common uses of attributes and parameters include simulating a system with many identical components (where the attribute is the ID number of a particular component) and simultaneous replications of a simulation run (where the attribute is the number of the replication). Attributes and parameters are generally used to reduce the visual size of a model without hindering its ability to depict large systems.

3 IPA IN THE EVENT GRAPH FRAMEWORK

In Freimer and Schruben (2001) we derived an IPA sample path derivative (with respect to a parameter of an edge delay time) for an event graph simulation. The derivation is similar to that given for GSMPs by Fu and Hu (1997) and Glasserman (1991), but modified to accommodate the structure of event graphs. The essential result is the following proposition: for all θ in a compact interval Θ and every $n \geq 0$, the sequence $\{(s_i, a_i, \Psi_i), i = 1, \ldots, n\}$ is a.s. constant in some neighborhood of θ. For conditions on the event graph under which this holds and a formal proof, see Freimer and Schruben (2001).

Taken with equation (1), the implication of this proposition is that epoch τ_j is the sum of a fixed sequence of delay times $X(a_i,I(i))$ in some neighborhood of θ. If we assume these delay times are differentiable in θ, the derivative of τ_j may be written:

$$\begin{aligned}\frac{d\tau_j(\theta)}{d\theta} &= \lim_{\Delta\theta \to 0} \frac{1}{\Delta\theta}\left(\tau_j(\theta+\Delta\theta) - \tau_j(\theta)\right) \\ &= \lim_{\Delta\theta \to 0} \frac{1}{\Delta\theta} \sum_{i \in \Psi_j} \left[X_{\theta+\Delta\theta}(a_i, I(i)) - X_\theta(a_i, I(i))\right] \\ &= \sum_{i \in \Psi_j} \frac{dX(a_i, I(i))}{d\theta} \quad \text{for } j = 1,\ldots,n \end{aligned} \tag{2}$$

As Suri (1987) and others have noted, these derivatives are easy to calculate over the course of the simulation run. When event a_i schedules some future event a_k, $k>i$, we have:

$$\frac{d\tau_k(\theta)}{d\theta} = \frac{d\tau_i(\theta)}{d\theta} + \frac{dX(a_k, I(k))}{d\theta}.$$

In the next section we will present an automated method for tracking these sums in an event graph using parameter passing.

We can now compute the derivative of the performance measure, $L(\theta)$. The usual construction of the performance measure is the general form:

$$L(\theta) = \int_0^{t_f} f(Z_t(\theta))dt, \tag{3}$$

where $Z_t(\theta)$ is the state of the process at time t, and t_f is a stopping time, the time at which the system state enters a pre-specified set of stopping values Φ_f. Note that this defi-

nition includes as special cases several common methods for terminating a simulation:

1. the overall number of events reaches a pre-specified value n_0;
2. the number of events of type a reaches a pre-specified value k;
3. a pre-specified termination time T is reached.

We can expand the state space to include a counter for the overall number of events or for the number of events of type a. Alternatively, we can add to the event graph a *Terminate* event, an edge from *Run* to *Terminate* with delay time T, and a state variable that counts the number of times *Terminate* is executed.

From performance measures of the type given in (3) we can construct a variety of other performance measures that may be of interest. For examples, see Fu and Hu (1997) and Glasserman (1991).

Now define $N(t)$ to be the number of events that have been executed up to and including time t. Since a basic assumption of event graphs (and GSMPs) is that the system state remains constant between events, we can write the performance measure as:

$$L(\theta) = \sum_{i=0}^{N(t_f)-1} f(s_i)[\tau_{i+1} - \tau_i].$$

Now consider the derivative of the performance measure $L(\theta)$ with respect to θ:

$$\frac{dL(\theta)}{d\theta} = \sum_{i=0}^{N(t_f)-1} f(s_i)\left[\frac{d\tau_{i+1}}{d\theta} - \frac{d\tau_i}{d\theta}\right].$$

Adapting some notation from Suri (1987), we define:

$$\Delta f_{s_i} = \begin{cases} f(s_{i-1}) - f(s_i) & \text{if } 0 < i < N(t_f) \\ f(s_{i-1}) & \text{if } i = N(t_f) \end{cases}. \qquad (4)$$

We now have:

$$\frac{dL(\theta)}{d\theta} = \sum_{i=1}^{N(t_f)} \Delta f_{s_i} \frac{d\tau_i}{d\theta}.$$

Substituting equation (2) we have:

$$\frac{dL(\theta)}{d\theta} = \sum_{i=1}^{N(t_f)} \Delta f_{s_i} \sum_{j \in \Psi_i} \frac{dX(a_j, I(j))}{d\theta}. \qquad (5)$$

We call this sum the IPA sample path derivative. In the next section we show how to modify an event graph so as to implement these calculations automatically.

4 IMPLEMENTING IPA FOR EVENT GRAPHS

Event graphs, with their visual representation of the relationships between events, present a natural framework for implementing IPA. Given an arbitrary event graph there is a straightforward method for implementing the calculations at the end of Section 3 using attributes and parameters. We first introduce a new state variable A, the accumulator, which at the end of the run contains the sum in equation (5), the IPA sample path derivative. Another state variable, G, will be used to pass delay time derivatives.

The algorithm for implementing IPA in an event graph is as follows:

1. Initialize variables A=0 and G=0 in the *Run* vertex.
2. Select an event vertex $a \in V$ from the event graph.
3. Define G as a parameter of event a. (Ignore this step for the *Run* vertex.)
4. Decide how to calculate Δf_a, the change in the function f when event a takes place. As is indicated by (4), this change may depend on the system state at the time a is executed. (Ignore this step for the *Run* vertex.)
5. To the set of state changes for event a add:
   Δf_a = (calculation determined in step 4)
   $A = A + \Delta f_a \cdot G$
   (Ignore this step for the *Run* vertex.)
6. For each edge e exiting vertex a, determine $dt_e/d\theta$, where t_e is the delay time along this edge. (If edge e leads to vertex b, $dt_e/d\theta$ is $dX(b,\cdot)/d\theta$.)
7. To each edge e exiting vertex a, add attribute value $G + \frac{dt_e}{d\theta}$.

Repeat steps 2-7 for each vertex in the set V.

The algorithm is represented graphically in Figure 5.

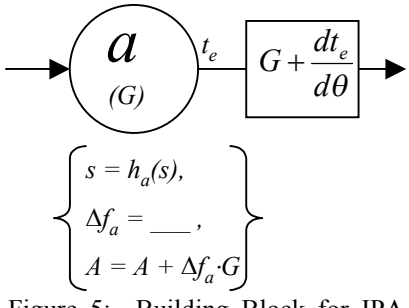

Figure 5: Building Block for IPA Implementation

As is indicated by (4), a slight modification must be made to handle the last event in the simulation. This may

be achieved by a post processing calculation, or in some cases by a direct modification of the event graph.

In the two-server queue example shown in Figure 3, Δf_a is determined by the event type of a:

$$\Delta f_a = \begin{cases} -1 & a \text{ is an } \textit{Arrive} \text{ event} \\ 1 & a \text{ is a } \textit{Start Service} \text{ event} \\ 0 & \text{otherwise} \end{cases}$$

Suppose this is an m/m/2 system, where θ is the mean service time, and we are generating service times via inverse transformation. In this case the only edge whose delay time is affected by a perturbation in θ is the one from *Start Service* to *Finish Service*, so

$$\frac{dt_e}{d\theta} = \begin{cases} \dfrac{t_e}{\theta} & \text{edge } e \text{ is from } \textit{Start Service} \text{ to } \textit{Finish Service} \\ 0 & \text{otherwise} \end{cases}$$

The resulting event graph for the single server queue is shown in Figure 6.

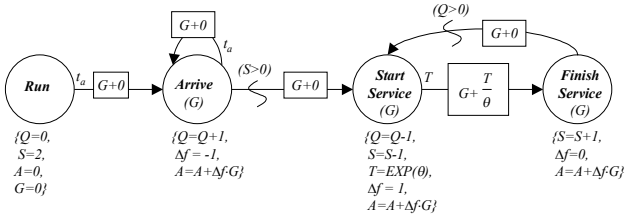

Figure 6: IPA Implementation for Multiple Server Queue

5 CONCLUSION

Since event graphs are a more flexible modeling framework than GSMPs, it is useful to be able to apply IPA results derived for GSMPs to event graphs. In this paper we have extended some of these results to event graphs, and described an automated implementation procedure.

ACKNOWLEDGMENTS

The research reported here was partially supported by a joint src (fj-490) and nsf (dmi-9713549) research project in semiconductor operations modeling.

REFERENCES

Freimer, M., and L. Schruben. 2001. Visualizing Infinitesimal Perturbation Analysis Estimators. Tech. Report # 1291. School of ORIE, Cornell University. Ithaca, NY. Available online via <http://www.orie.cornell.edu>.

Fu, M., and J. Hu. 1997. *Conditional Monte Carlo: Gradient Estimation and Optimization Applications*. Boston, MA: Kluwer.

Glasserman, P. 1991. *Gradient Estimation via Perturbation Analysis*. Boston, MA: Kluwer.

Schruben, L. 1983. Simulation Modeling with Event Graphs. *Communications of the ACM* 26 (11): 957–963.

Schruben, L. and E. Savage. 1996. Visualizing Generalized Semi-Markov Processes. In *Proceedings of the 1996 Winter Simulation Conference*, ed. J. M. Charnes, D. M. Morrice, D. T. Brunner, and J. J. Swain, 1465–1470. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers.

Som, T., and R. Sargent. 1989. A Formal Development of Event Graphs as an Aid to Structured and Efficient Simulation Programs. *ORSA Journal on Computing* 1 (2): 107–125.

Suri, R. 1987. Infinitesimal Perturbation Analysis for General Discrete Event Systems. *Journal of the Association for Computing Machinery* 34 (3): 686–717.

AUTHOR BIOGRAPHIES

MICHAEL FREIMER is a Ph.D. candidate at Cornell University's School of Operations Research and Industrial Engineering. He received his undergraduate degree in mathematics from Harvard. Prior to attending graduate school he worked for an operations research consulting firm, Applied Decision Analysis, Inc. in Menlo Park, CA. His research interests are in simulation modeling and revenue management. His email address is <mfreimer@orie.cornell.edu>.

LEE SCHRUBEN is a Professor in the Department of Industrial Engineering and Operations at the University of California at Berkeley. His research interests are in statistical design and analysis of simulation experiments and in graphical simulation modeling methods. His simulation application experiences and interests include semiconductor manufacturing, dairy and food science, health care, banking, and the hospitality industry. His email address is <schruben@ieor.berkeley.edu>.