Visualizing Infinitesimal Perturbation Analysis Estimators

Mike Freimer, ORIE, Cornell University
Lee Schruben, IEOR, UC Berkeley

May 15, 2000
(Revised: June 14, 2001)

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.

1.0 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 q 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 q. Such gradients may be required in a variety of contexts, such as stochastic optimization and output sensitivity analysis. For examples, see [1] and [3].

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 q; runs are made with parameter values q and $(q+Dq)$ for some suitably small Dq. If $L(w,q)$ is a realization of the system performance measure with parameter value q, the finite differencing gradient estimate is $\frac{1}{\Delta q}\left[L(w_1, q+\Delta q) - L(w_2, q)\right]$. 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 $W(q)$ of customers in a single server queue with mean service time q. 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.

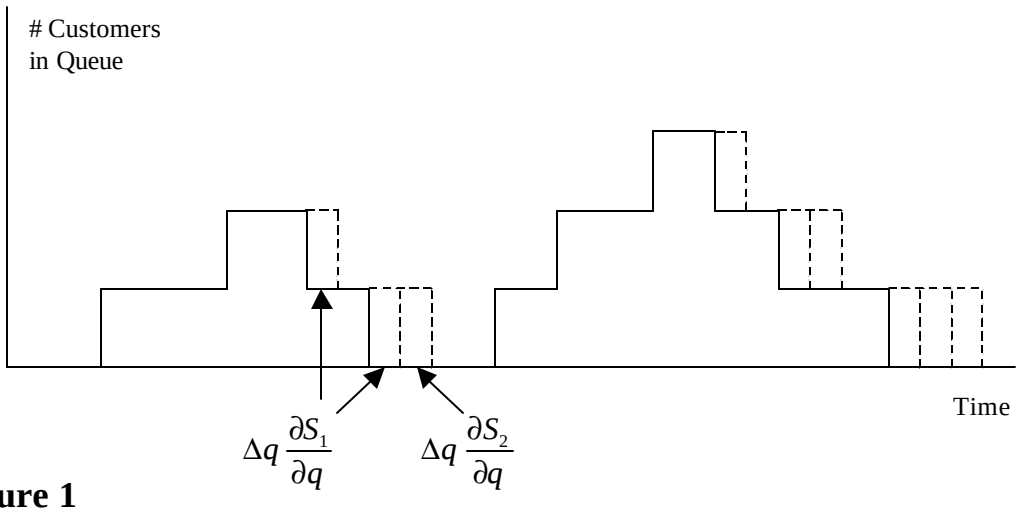

Figure 1
Sample realization for single server queue

Let $S_i(w,q)$ be the service time of the i^{th} customer. A small increase Dq in q will cause individual service times to increase by an amount $\Delta q \frac{\partial S_i(w,q)}{\partial q} + o(\Delta q)$. 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 customer is only affected by the service times of customers who come before him in the same busy period. For example, if he is the 3rd customer in the busy period, his waiting time is lengthened by the increases in service time of the two customers ahead of him. In general, the waiting time of a customer is increased by the sum of the increases in service time of the customers ahead of him in the busy period.

If the change Dq 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 customers as follows. Suppose the simulation is run for M busy periods where the m^{th} busy period starts with customer k_m. The change in overall waiting time due to Dq is $\sum_{m=1}^{M}\sum_{i=k_m}^{k_{m+1}-1}\sum_{j=k_m}^{i}\Delta q \frac{\partial S_j(w,q)}{\partial q} = \sum_{m=1}^{M}\sum_{j=k_m}^{k_{m+1}-1}(k_{m+1}-j)\Delta q \frac{\partial S_j(w,q)}{\partial q}$. 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 $W(q)$ we let Dq go to zero, so the order of events in the simulation run remains fixed. The sample path derivative can then be written as:

$$\begin{aligned}\frac{dW(q,w)}{dq} &= \lim_{\Delta q \to 0}\frac{1}{\Delta q}\left[\frac{1}{N}\sum_{m=1}^{M}\sum_{j=k_m}^{k_{m+1}-1}(k_{m+1}-j)\Delta q \frac{\partial S_j(w,q)}{\partial q}\right] \\ &= \frac{1}{N}\sum_{m=1}^{M}\sum_{j=k_m}^{k_{m+1}-1}(k_{m+1}-j)\frac{\partial S_j(w,q)}{\partial q}\end{aligned} \qquad \textbf{(1.1)}$$

where N is the number of customers served during the simulation run. So in order to compute $\frac{dW(q,w)}{dq}$, we do not choose a particular value for Dq; 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, q, and of a random number, $U(w)$. For example, in the m/m/1 queue, where q is the mean service time, we have $S(w,q) = F_S^{-1}(U(w),q) = -q\ln(1-U(w))$, and $\frac{\partial S(w,q)}{\partial q} = -\ln(1-U(w)) = \frac{1}{q}S(w,q)$. 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. It is also a reason that Suri and others have argued allows us to

implement IPA not just for simulated systems, but for physical, real-time systems as well. See, for example, [12].

The IPA derivative calculated in (1.1) is a sample path derivative. In other words, it is the derivative with respect to q 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, $\frac{\partial E_w[W(w,q)]}{\partial q}$. A central question is whether the IPA sample path derivative is an unbiased estimator. In other words, under what conditions is it true that

$$E_w\left[\frac{\partial W(w,q)}{\partial q}\right] = \frac{\partial E_w[W(w,q)]}{\partial q} \quad ? \qquad \textbf{(1.2)}$$

The conditions for unbiasedness and many other useful IPA results are often derived in the framework of generalized semi-Markov processes (GSMPs) [1] [3]. 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 [7] specifically to facilitate model building for discrete-event systems. Event graphs are a general modeling paradigm that includes GSMPs as a subset. Savage and Schruben [9] 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.

Event graphs, with their visual representation of the relationships between events, present a natural framework for implementing IPA. Our purpose in this paper is to describe how this implementation can be easily accomplished, and how the pictorial nature of event graphs can lend insight into IPA calculations. In Section 2 we review the formal definitions of GSMPs and event graphs and discuss the differences between the two. In Section 3 we describe the implementation of IPA for event graphs by deriving the appropriate calculations and giving an automatic procedure for implementation. In Section 4 we describe conditions on the event graph under which the IPA estimator is unbiased, i.e. under which equation (1.2) holds. In Section 5 we describe an extended example with the intention of showing how conditional IPA calculations can be implemented for event graphs.

2.1 The GSMP Framework

A generalized semi-Markov process (GSMP) model is the common framework for which IPA results are derived. We construct GSMPs following the developments of Fu and Hu [1], and Glasserman [3]. Define the following:

S	countable set of states
E	finite set of events
a, b	generic events in E
$E(s)$	non-empty set of active events in state $s \hat{I} S$
$p(s'; s, a)$	probability of jumping from state s to s' when event $a \hat{I} E(s)$ occurs
$F_a(.)$	distribution of time interval until event $a \hat{I} E$ occurs.

The process begins in a pre-specified state $s_0 \hat{I} S$ and evolves as follows. The state remains constant until the occurrence of an event in $E(s_0)$, at which time it may change. (The set $E(s)$, $s \hat{I} S$, contains the *active* events for state s, i.e. those that are scheduled to occur when the system is in state s.) Associated with each event in E is an event clock, which displays the time until the next occurrence of the event. We assume that all clocks run down at the same unit rate. The initial clock value for each $b \hat{I} E(s_0)$ is determined according to $F_b(.)$. The clock value for any event not in $E(s_0)$ is set to 0.

The first event to occur (call it a) is the event in $E(s_0)$ with the smallest positive clock value. When this event takes place, the state changes to new state $s_1 \hat{I} S$ with probability $p(s_1; s_0, a)$. Clocks for events that were active in state s_0 and that are also active in s_1 are reduced appropriately. The clock for any event b that is newly active in state s_1 is set according to $F_b(.)$, and the clock for any event that was active in s_0 but is no longer active in s_1 is set to 0. The state now remains fixed at s_1 until the occurrence of the next event: the one with the smallest positive clock value. The process continues in a similar way, proceeding from state to state.

As an example, consider an m/m/1 queue ([1] [3]). The set of states $S=\{0,1,2,\ldots\}$ represents the number of customers in the system. The set of events, E, consists of arrivals, a, and service completions, b. We have $E(0) = \{a\}$, $E(s) = \{a, b\}$ for $s \geq 1$, $p(s+1; s, a) = 1$ for $s \geq 0$, and $p(s-1; s, b) = 1$ for $s \geq 1$. Finally $F_a(.)$ and $F_b(.)$ are the CDF's of the inter-arrival and service completion times respectively.

2.2 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.

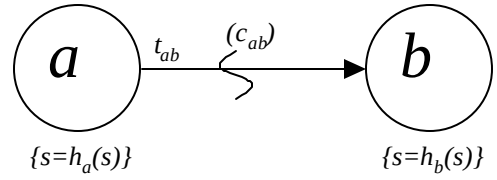

Figure 2
Event graph building block

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.

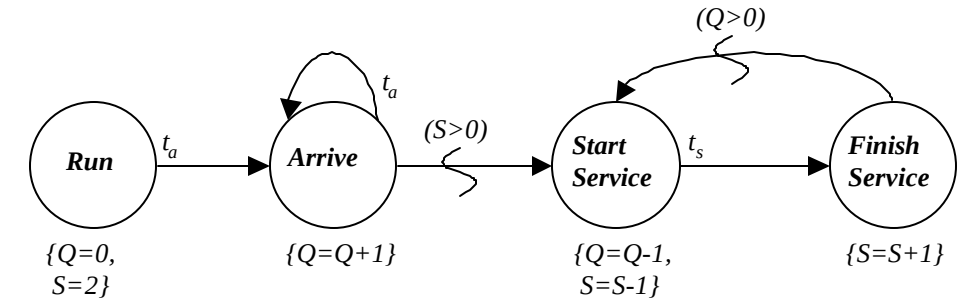

Figure 3
Event graph: two servers, single queue

The evolution of an event graph model is similar in spirit to that of the GSMP described in section 2.1. 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 proofs 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. For example, consider the event graph shown in Figure 4. After event *a* is executed, both *b* and *c* will appear on the FEL at the same time. Since they affect the system state differently, we must specify which event should be executed first. If *b* should be executed before *c*, we say *b* has a higher *priority* than *c*. We assume the priority relationships among events satisfy a transitive property.

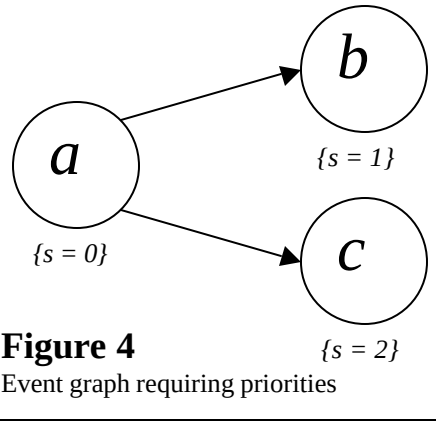

Figure 4
Event graph requiring priorities

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
t_n	epoch of the n^{th} event to occur ($t_0 \equiv 0$)
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 $\{(t_i, s_i), i = 0,\ldots,n; (a_i), i = 1,\ldots,n\}$. The clock sample $X(a,n)$ refers to the edge delay time $t_{\cdot,a}$ used to schedule the n^{th} instance of event a. Clock samples $X(a,i)$ and $X(a,j)$ may be sampled from different distributions if $i \neq j$. For example, consider the fragment of an event graph represented by Figure 5.

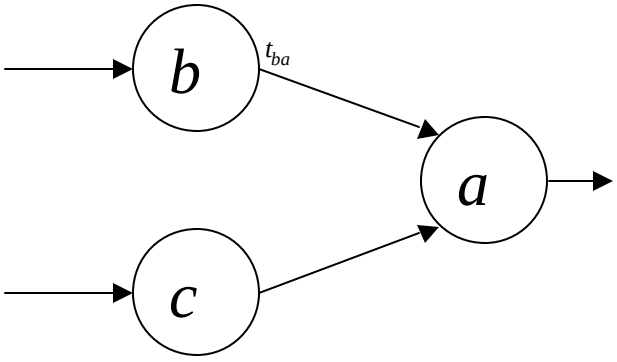

Figure 5
Event graph with multiple events that may schedule "a"

Suppose the 3rd instance of event a is scheduled by event b, and the 4th instance of event a is scheduled by event c. Then $X(a,3)$ will be a random variable distributed according to, say, $F_{ba}(.)$, and $X(a,4) \equiv 0$.

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 $I(n) \circ I(a_n,N(a_n, n))$.

We say that the i^{th} event is *triggered* or *scheduled* by a previous event numbered $k(i)$ if the event graph contains a directed edge from $a_{k(i)}$ to a_i that caused the i^{th} event to be added to the FEL at time $t_{k(i)}$. Adapting Fu and Hu's notation [1], the triggering event

index set Ψ_i of a_i is defined recursively: $\Psi_i = \{k(i)\} \cup \Psi_{k(i)}$ if $i>1$, and $\Psi_i = f$ if $i=1$. The set Ψ_i contains the genealogy of event a_i. Using this notation we have:

$$t_n = \sum_{i \in \Psi_n} X(a_i, I(i)) \qquad \textbf{(2.1)}$$

for $n \geq 1$.

2.2.1 Enrichments to Event Graphs

An enrichment of event graphs is the use of *canceling* edges. If the condition of a canceling edge holds, then the first (if any) occurrence of the destination event is removed from the FEL. For example, if a model includes machine breakdowns, the completion of work on an item will have to be canceled if a breakdown occurs. In the event graph the canceling edge is represented as a dashed arrow.

Another enrichment is 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 6. 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.

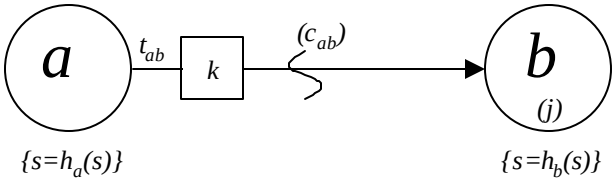

Figure 6
Event graph building block with parameters and attributes

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.

2.3 Comparing GSMPs and Event Graphs

Next we discuss the differences between GSMPs and event graphs. Savage and Schruben [9] have shown that there is a direct translation from GSMPs to event graphs; any GSMP can be modeled as an event graph. However it is easy to find an event graph that cannot be translated directly to a GSMP. For example, Savage and Schruben discuss a collision-free bus network model that is not a GSMP. (This model was originally described by Iglehart and Shedler [5].) Another simple example is shown in Figure 7:

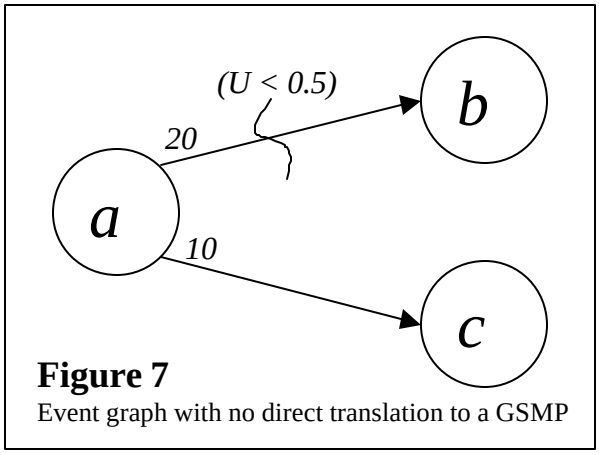

Figure 7
Event graph with no direct translation to a GSMP

Here *U* is a function that returns a random number uniformly distributed on [0,1]. When event *a* occurs, event *c* is added to the FEL after a delay of 10 time units. Event *b* appears on the FEL after a delay of 20 time units with probability one half. Unless we expand the set of state variables to include an indicator for *{U<0.5}*, at time 5 we cannot determine which events are active (scheduled on the FEL) from the system state. Unfortunately this is an important feature of the GSMP framework; the system state determines which events are active. This example illustrates the most significant distinction between event graphs and GSMPs.

Another feature of the generic GSMP framework is that there can be no more than one active event of a given type. For example, a GSMP representation of a multiple-server queue will have distinct *Finish Service* events for each server. The event graph model shown in Figure 3 has a single *Finish Service* event, which can appear twice on the FEL if both servers are busy. This precludes a direct translation of this event graph to the generic GSMP representation.

These two distinctions do not stand in the way of extending the IPA results to event graphs. As we shall see in the next section, the derivation of the IPA estimator does not refer to the mechanism by which the system state and the set of active events are updated, or to the number of active events of each type. Under assumptions similar to the ones made for GSMPs [3], we can show an IPA derivative calculation for event graphs.

However there is a third distinction that we might expect to be a problem. A primary assumption of IPA is that there is zero probability of two events occurring simultaneously. To achieve this in the generic GSMP framework, one typically assumes that the distributions on the clock times are continuous. For example, Glasserman's assumption is stated [3]:

(A1') For each $q \hat{I} Q$ and each $a \in E$, $F_a(x,q)$ is continuous in x and zero at $x=0$,

for a compact interval Q. However in an event graph the time delay between events is often zero; two events are purposefully scheduled for the same time. For example, Figure 3 has two edges with zero delay times. Zero delay times turn out not to be a difficulty; we will show that as long as we rule out other cases of simultaneous events, a sample path derivative exists and can still be calculated from a single simulation run.

3.1 IPA in the Event Graph Framework

The purpose of this section is to derive, given some basic assumptions, a sample path derivative for an event graph simulation. Others have pursued a similar goal with respect to Petri nets. Since Petri nets and GSMPs have been shown to have the same modeling power, IPA results developed for GSMPs may be applied to Petri nets. (See, for example, [4] and [13].) Unfortunately the inclusion of zero-timed edges precludes this approach for event graphs. We present derivations structurally similar to [1] and [3], but modified to account for constant and zero-timed edges. A straightforward method for the automatic implementation of IPA on an event graph will be given in the next section.

We derive the IPA derivative calculations for event graphs directly, although one could use a more roundabout approach. Typically an event graph G can be re-drawn as a graph G' having only non-zero edge delay times. (Zero-timed edges are used to make the logical structure of the event graph more transparent. One can usually eliminate a zero-timed edge by combining event nodes and embedding the edge logic in the state change functions.) The two graphs G and G' are *behaviorally equivalent;* given the same sequence of input random numbers, the redrawn graph G' will return the same performance measure as G [10]. By applying IPA to G', one can find the sample path derivative for G. The trouble with this approach is that some effort is required to redraw the graph; we will show that IPA is applicable directly to the original graph.

For the remainder of this section we restrict ourselves to the case where q is a parameter of the distribution of an event delay time; this means that q is a parameter of $F_e(.)$ for some edge $e \in D$. We do not consider the case where q is a routing parameter, a parameter of the edge condition c_e for some $e \in D$. Fu and Hu do analyze this case, in the context of conditional IPA for GSMPs [1]. However in this case the standard IPA derivative with respect to q is zero; only the derivative conditioned on a change in the event sequence $\{a_1, a_2, \ldots\}$ due to a perturbation in q is non-zero. In an event graph one might try to get around this by modeling the routing probability with edge delay times. For example, suppose that the routing probability p is a rational number, x/y. The event

graph shown in Figure 8 will execute event *b* with probability *p* and event *c* with probability (1-*p*). (The function *Exp(x)* returns an exponentially distributed random variable with mean *x*. Also note the use of canceling edges between events *b* and *c*.)

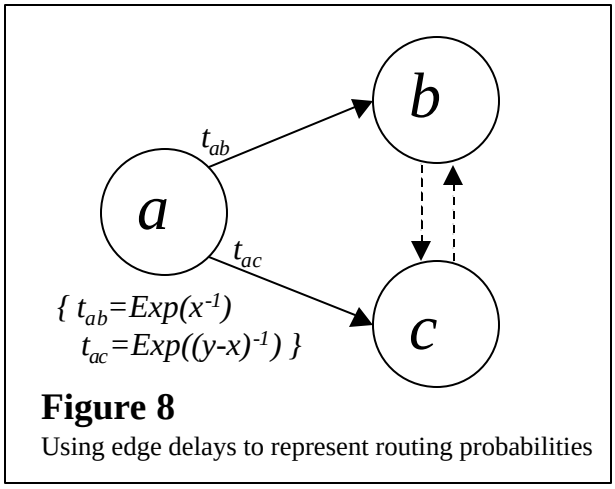

Figure 8
Using edge delays to represent routing probabilities

Of course the trouble with this approach is that it disrupts the timing of events in the simulation. Suppose the graph shown in Figure 8 were embedded in a larger event graph of a queuing system, representing a customer's choice between servers *b* and *c*. Suppose also that during the simulation run a customer chooses server *b*. Increasing *x* by a small amount *Dx* then decreases the average waiting time in the system, but only because the customer "waits" a little less time to make his decision, not because server *b* becomes more likely to be chosen. On the other hand, if one considers a performance measure that is not time-based (e.g. the fraction of customers choosing server *b* over server *c*), the IPA sample path derivative will still be zero.

Once again, we restrict ourselves to the case where *q* is a parameter of the distribution of an edge delay time. We first describe more formally what is meant by a performance measure. The usual construction ([3] [12]) is to allow performance measures having the general form:

$$L(q) = \int_0^{t_f} f(Z_t(q))dt\,, \qquad \textbf{(3.1)}$$

where $Z_t(q)$ 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 definition 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 (3.1) we can construct a variety of other performance measures that may be of interest. For example, consider a G/G/1 queue. Let $f_1(Z_t(q))$ be the number of customers in queue at time t, and define $L_1(q) = \int_0^{t_f} f_1(Z_t(q))dt$. Let $f_2(Z_t(q))=1$, and define $L_2(q) = \int_0^{t_f} f_2(Z_t(q))dt$. Then $L_3(q) = L_1(q)/L_2(q)$ is the average number of waiting customers during the interval $[0,t_f]$. If t_f is the departure time of the k^{th} customer, then $L_4(q) = k/L_2(q)$ is the throughput rate of the queue, and $L_5(q) = L_1(q)/k$ is the average waiting time. The derivatives of $L_3(q)$, $L_4(q)$, and $L_5(q)$ can be calculated from $\frac{dL_1(q)}{dq}$ and $\frac{dL_2(q)}{dq}$ via the chain rule.

Next we describe some assumptions required for the sample path derivative to exist and be easy to calculate. First re-write assumption (A1') to allow for zero-timed edges:

(A1) For each $q \hat{I} Q$ and each $e \hat{I} D$, either: (i) $F_e(x,q)$ is one for $x \geq 0$ and zero otherwise, or (ii) $F_e(x,q)$ is continuous in x and zero at $x=0$.

As was illustrated by the example given in the introduction, the IPA derivative calculation is essentially a sum of delay time derivatives. This requires that the delay times be continuously differentiable in q, and that the derivative of a delay time be a function of the delay time itself:

(A2) For every $a \in V$ and $k \geq 1$, $X_q(a,k)$ is, with probability one, a continuously differentiable function of q on Θ, and $dX_q(a,k)/dq$ is a measurable function of $X_q(a,k)$.

Together with (A1), the next assumption is sufficient to rule out simultaneous events for a fixed q w.p.1, except for those purposefully scheduled by zero-timed edges.

(A3) The clock samples $X(a,i)$, $a \in V$, $i=1,2,\ldots$ are all independent of one another.

Merely ensuring that all random variables are generated using independent random numbers is not sufficient to satisfy (A3). We must rule out cases like the one shown in Figure 9, where the event graph satisfies (A1), but events *b* and *c* are scheduled for the same time.

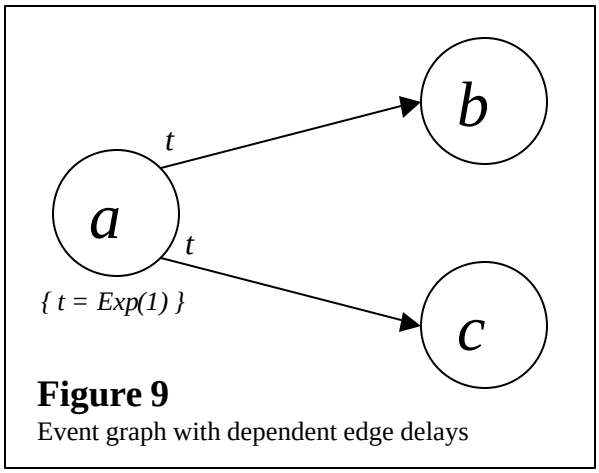

Figure 9
Event graph with dependent edge delays

The next two assumptions guarantee that as long as events do not change order, the system state and triggering event index sets Ψ_n remain fixed.

(A4) The system state contains no reference to the simulation clock. The edge conditions c_e, $e \in D$, and state transition functions $h_a(.)$, $a \in V$, are functions only of the current system state.

The final assumption ensures that the execution priorities between events connected by zero-timed edges are clear. We borrow a term from Som and Sargent [11] and define the *closure* of vertex $a \in V$ to be $\{a\} \cup$ set of all vertices accessible from vertex a through a path which does not have any time delay on it.

(A5) For every vertex $a \in V$, the execution priorities among the events in the closure of a are well specified and do not depend on the simulation clock.

Proposition 1
Under assumptions (A1), (A3), (A4), and (A5), for all $q \in \Theta$ 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 q.

The intuition for the argument is that an event a and its closure essentially act as a single meta-event. Suppose that in the course of the simulation run the execution of event a is delayed due to a perturbation in parameter q. All of the events in the closure of a are likewise delayed. However because the system state prior to a remains constant, by assumptions (A4) and (A5) the sequence and effects of the events in a's closure also remain fixed.

Proof
Again following Som and Sargent [11], we define a *computational trajectory*. Given a sample path $\{(t_i, s_i), i = 0,\ldots,n; (a_i), i = 1,\ldots,n\}$, there is a unique subset $\{j_0, \ldots, j_k\}$ of the indices $\{0, \ldots, n\}$ such that:

1) $k+1$ is the number of unique elements of $\{t_0, \ldots, t_n\}$;
2) $j_k = n$;
3) for $i = 0, \ldots, k-1$: j_i is the largest integer such that $t_{j_i} < t_{j_{i+1}}$.

By this definition $t_{j_0} = 0$. This is called a computational trajectory since a performance measure of the form given in (3.1) is a function of only $\{(t_{j_i}, s_{j_i}): i = 0,\ldots,k\}$. We present an induction argument on the computational trajectory.

We have assumed that the initial FEL contains only one event, the *Run* event scheduled at time 0. In this case j_0 is index of the last event that occurs at time $t_0=0$, and all the events up to the j_0^{th} are unaffected by any perturbation in parameter q. Therefore all (s_i, a_i, Ψ_i), $i \le j_0$ are a.s. fixed for a neighborhood of q.

Now for the induction step. Suppose that for $m \ge 0$, all (s_i, a_i, Ψ_i), $i \le j_m$ are a.s. fixed in a neighborhood of q. Consider any event a_p, $p \le j_m$ that occurs up to the j_m^{th} event. Since by assumption s_{p-1} is a.s. constant in a neighborhood of q, by assumption (A4) the conditions on each of the edges leading from event vertex a_p remain constant at time t_p for the same neighborhood of q. Therefore the set of events scheduled by event a_p is also fixed. Since this is true for all $p \le j_m$, the set of events on the FEL just after time t_{j_m} is a.s. constant for this neighborhood of q. By (A1) and (A3), w.p.1 none of these events are scheduled for the same time, and their scheduled order of execution is constant for a neighborhood of q. Therefore $(a_{j_m+1}, \Psi_{j_m+1}, s_{j_m+1})$ is a.s. constant for a neighborhood of q.

If the closure of a_{j_m+1} is $\{a_{j_m+1}\}$, then $j_{m+1}=j_m+1$, and we are done. Otherwise there will be at least one other event that takes place at t_{j_m+1}. All of the events that occur at t_{j_m+1} will be in the closure of a_{j_m+1} and have a_{j_m+1} in their triggering event index sets. Because s_{j_m} is a.s. constant in a neighborhood of q, by (A4) and (A5) the event sequence that occurs at t_{j_m+1} also remains constant in this neighborhood. Likewise, the sequence of states and triggering event index sets at t_{j_m+1} remain constant in this neighborhood w.p.1. Since the last event to take place at t_{j_m+1} is the j_{m+1}^{th}, the sequence (s_i, a_i, Ψ_i), $j_m \le i \le j_{m+1}$ is a.s. constant in a neighborhood of q. $\otimes$

It is not hard to extend assumption (A1) and Proposition 1 to allow for non-zero constant edge delay times that do not depend on q. Constant delay times that do depend on q may be a problem, as is shown by the event graph fragment in Figure 10.

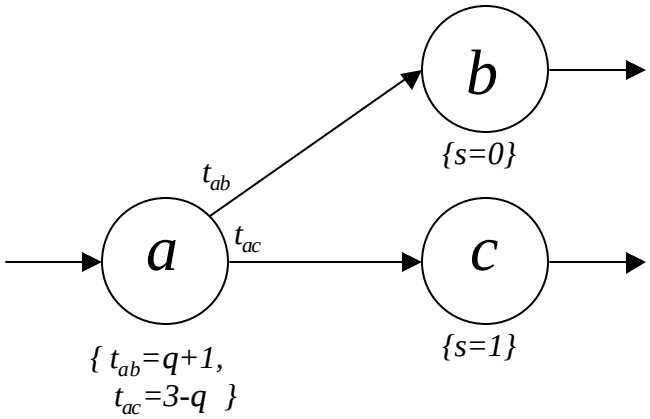

Figure 10
Event graph with constant delay times

For this example Proposition 1 fails to hold at $q=1$.

Since by equation (2.1), $t_j = \sum_{i \in \Psi_j} X(a_i, I(i))$ for $j=1,\ldots,n$, the implication of Proposition 1 is that epoch t_j is the sum of a fixed sequence of delay times $X(a_i,I(i))$ in some neighborhood of q. By (A2) these delay times are differentiable in q, so the derivative of t_j may be written:

$$\begin{aligned}\frac{dt_j(q)}{dq} &= \lim_{\Delta q \to 0} \frac{1}{\Delta q}\left(t_j(q+\Delta q) - t_j(q)\right) \\ &= \lim_{\Delta q \to 0} \frac{1}{\Delta q} \sum_{i \in \Psi_j} \left[X_{q+\Delta q}(a_i, I(i)) - X_q(a_i, I(i))\right] \\ &= \sum_{i \in \Psi_j} \frac{dX(a_i, I(i))}{dq} \qquad (3.2)\end{aligned}$$

for j = 1, …, n.

As Suri [12] 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:

$$\frac{dt_k(q)}{dq} = \frac{dt_i(q)}{dq} + \frac{dX(a_k, I(k))}{dq}.$$

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

We are now ready to compute the derivative of the performance measure, $L(q)$. 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:

$$L(q) = \sum_{i=0}^{N(t_f)-1} f(s_i)[t_{i+1} - t_i]. \tag{3.3}$$

Now consider the derivative of the performance measure *L(q)*. Since q is a parameter of an edge delay time, we have:

$$\frac{dL(q)}{dq} = \sum_{i=0}^{N(t_f)-1} f(s_i)\left[\frac{dt_{i+1}}{dq} - \frac{dt_i}{dq}\right].$$

Adapting some notation from Suri [12], we define:

$$\Delta f_{s_i} = \begin{cases} f(s_{i-1}) - f(s_i) & \text{if } 0 < i < N(t_f) \\ f(s_{i-1}) & \text{if } i = N(t_f) \end{cases}. \tag{3.4}$$

We now have:

$$\frac{dL(q)}{dq} = \sum_{i=1}^{N(t_f)} \Delta f_{s_i} \frac{dt_i}{dq}.$$

Substituting equation (3.2) we have:

$$\frac{dL(q)}{dq} = \sum_{i=1}^{N(t_f)} \Delta f_{s_i} \sum_{j \in \Psi_i} \frac{dX(a_j, I(j))}{dq}. \tag{3.5}$$

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.

3.2 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 3.1 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 (3.5), 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 (3.4), 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)
   $A = A + \Delta f_a \times G$
   (Ignore this step for the *Run* vertex.)
6. For each edge e exiting vertex a, determine dt_e/dq, where t_e is the delay time along this edge. (If edge e leads to vertex b, dt_e/dq is $dX(b,\cdot)/dq$.)
7. To each edge e exiting vertex a, add attribute value $G + \frac{dt_e}{dq}$.

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

The algorithm is represented graphically in Figure 11.

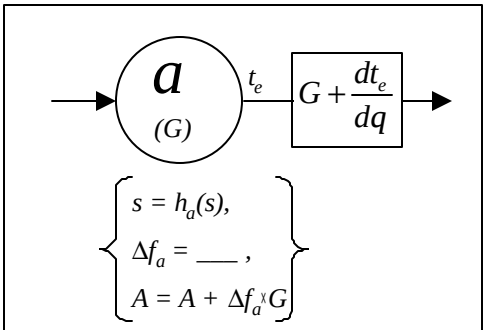

Figure 11
Building block for IPA implementation

As is indicated by (3.4), 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 q is the mean service time. In this case the only edge whose delay time is affected by a perturbation in q is the one from *Start Service* to *Finish Service*, so

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

The resulting event graph for the single server queue is shown in Figure 12. The graph has some redundancy, but this is easily eliminated.

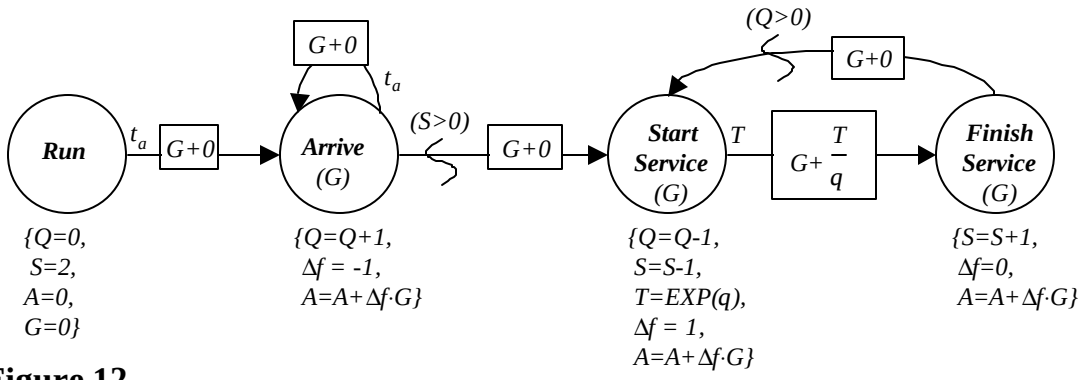

Figure 12
IPA implementation for multiple server queue

4.0 Conditions for Unbiasedness

We next turn to the issue of unbiasedness, in other words, the conditions under which is it true that:

$$E_w\left[\frac{\partial L(w,q)}{\partial q}\right] = \frac{\partial E_w[L(w,q)]}{\partial q}. \qquad \textbf{(4.1)}$$

Glasserman [3] gives conditions under which the IPA estimator for a GSMP is unbiased. The primary condition is the *commuting condition*. The essence of the commuting condition is that when a perturbation in the parameter q causes two events to occur simultaneously, the resulting system state and set of active (scheduled) events are the same regardless of the order in which the two events are executed. This guarantees that the remainder of the sample path is also independent of the order of execution. Jacobson and Yücesan have pointed out that the general problem of checking whether there exists a sequence of events such that if the last two events are interchanged, two distinct states are reached, is NP-hard [6]. However as we shall see, it is possible to find sufficient conditions on the event graph structure for the commuting condition to hold.

Since for a GSMP the set of active events is determined by the system state, the commuting condition can be written in terms of the system state only. For an event graph the situation is more complicated. First, the system state and FEL are not directly tied to one another, so the commuting condition must be described in terms of both the state variables and the FEL. More significantly, our construction of event graphs has allowed zero-timed edges, so the simultaneous occurrence of two events may cause additional interactions between the events' closures.

The intuition behind the commuting condition is related to the notion of event interaction described by Som and Sargent [11]. For a given pair of events, they provide a set of

sufficient conditions under which the execution priority between the two events is irrelevant. More specifically, suppose two events *a* and *b* are scheduled to occur simultaneously at some future time t_{ab}. In Som and Sargent's parlance, events *a* and *b* do not *interact* if:

(i) no event in the closure of *a* causes event *b* to be cancelled, and vice-versa;
(ii) the system state and FEL at the end of time t_{ab} (after the last event at time t_{ab} has been executed) do not depend on whether *a* or *b* was executed first.

When a pair of interacting events can be scheduled at the same time, it is necessary to provide an execution priority. Som and Sargent provide sufficient conditions on the structure of the event graph under which a given pair of events do not interact.

A candidate for the commuting condition in the framework of event graphs is therefore:

(A6') For all $a,b \in V$, *a* and *b* do not interact.

Under this condition any pair of events that can be scheduled to occur simultaneously can be executed in either order. Unfortunately (A6') is too restrictive; it rules out *all* event graphs for which event priorities are necessary. For example, consider the single server queue shown in Figure 13.

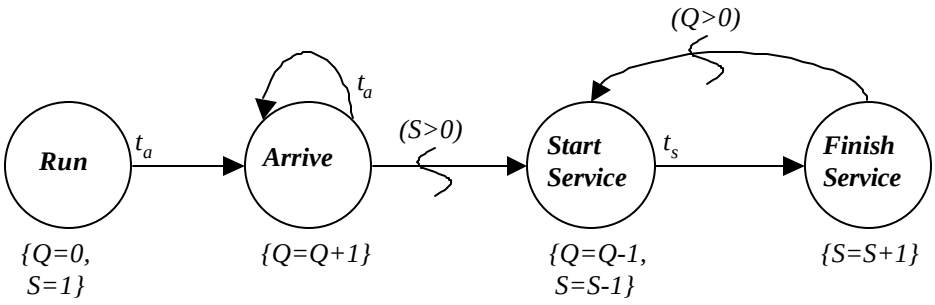

Figure 13
Event graph: single server queue

It turns out that the sample path derivative for the average waiting time in this model is an unbiased estimate of the derivative of the expected waiting time. However the *Arrive* and *Start Service* events require an execution priority to be specified. To see this, suppose there is a customer waiting *(Q>0)*, the server is busy *(S=0)*, and the *Arrive* and *Finish Service* events happen to be scheduled simultaneously. If the *Finish Service* event is executed first, the server is freed *(S=1)*, and since there is a customer waiting *(Q>0)*, a *Start Service* event is placed on the FEL without delay. Now the *Arrive* and *Start Service* events are scheduled simultaneously. If the *Arrive* event is executed next, the queue is incremented *(Q=Q+1)*, and since the server is free *(S=1)*, a *Start Service* event is placed on the FEL. Now there are *two* Start Service events on the FEL, and we have created a "phantom" server. To avoid this problem we must give the *Start Service* event a higher

priority than the *Arrive* event. Therefore the *Arrive* and *Start Service* events interact, and the event graph in Figure 13 fails condition (A6').

To arrive at a milder version of condition (A6'), we start with a definition: for a given event graph, let $H = \{b \in V: \exists\ a \in V,\ e_{ab} \in D,$ and $t_{ab} > 0\}$. In other words, H is the set of vertices having entering edges with non-zero delay times. (For the event graph shown in Figure 13, $H = \{Arrive, Finish\ Service\}$.) The new condition is:

(A6) For all $a,b \in H$, a and b do not interact.

The idea is to partition the sample path into *meta-events*, groups of events that occur simultaneously and that schedule one another directly. A meta-event will consist of an event in the set H and some other events from its closure. We allow priorities to be specified among the events in the closure, but not for the initial event from H.

To develop this idea more formally, we begin with the sample path $\{(t_i, s_i),\ i = 0,\ldots,n;\ (a_i),\ i = 1,\ldots,n\}$. Partition the events $a_1, \ldots, a_n$ into *meta-events*:

$\forall i,j$: a_i and a_j are in the same meta-event if $t_i = t_j$ and either $i\ \hat{I}\ \Psi_j$ or $j\ \hat{I}\ \Psi_i$.

This defines an equivalence relationship on the set of events in the sample path. Consider an arbitrary meta-event whose component events are arranged by order of execution: $\{a_{h_1}, \ldots, a_{h_m}\}$. We call the first event of a meta-event the *lead* event. Since the lead event a_{h_1} must have been scheduled at some time prior to t_{h_1}, we have $a_{h_1} \in H$.

If there are k meta-events in the sample path, order them chronologically by lead event and label them $\bar{a}_1, \ldots, \bar{a}_k$. For $i \in \{1,\ldots,k\}$ let $\bar{t}_i$ be the time at which the events in $\bar{a}_i$ take place, and let $\bar{s}_i$ be the system state immediately after the last event in $\bar{a}_i$ is executed. If we define $\bar{s}_0 \equiv s_0$ and $\bar{t}_0 \equiv t_0$, we can now re-write $L(q)$ as:

$$L(q) = \sum_{i=0}^{k-1} f(\bar{s}_i)(\bar{t}_{i+1} - \bar{t}_i). \qquad \textbf{(4.2)}$$

The following propositions are Glasserman's Lemma 3.1, Theorem 3.1, and Theorem 3.2 applied to event graphs [3]. We omit the full proofs since they are essentially Glasserman's arguments applied to meta-events.

Proposition 2
Under assumptions (A1) - (A6),

i) Every $\bar{t}_i$ is a.s. continuous in θ throughout Θ.

ii) At a discontinuity of $\bar{s}_i$, $\bar{t}_{i+1} = \bar{t}_i$.

The proof is based on the fact that for each m, $\bar{t}_1,\ldots,\bar{t}_m$ and $\bar{s}_1,\ldots,\bar{s}_m$ are continuous in q whenever $\bar{a}_1,\ldots,\bar{a}_m$ are, i.e. whenever changing q by a small increment does not change the first m meta-events. The idea is to look at the first discontinuous meta-event, $\bar{a}_j$, and show that $\bar{t}_j$ is continuous, even though $\bar{a}_j$ is not. Furthermore $\bar{a}_j$ can be discontinuous only if there is another meta-event $\bar{a}_{j+1}$ that takes place at the same time; a small perturbation in q causes $\bar{a}_j$ and $\bar{a}_{j+1}$ to change order. The commuting condition ensures that the sample path after $\bar{t}_{j+1}$ is independent of the order of $\bar{a}_j$ and $\bar{a}_{j+1}$, so the epochs of the meta-events following $\bar{a}_{j+1}$ are continuous up until the next discontinuous meta-event $\bar{a}_{j'}$. At j' we may repeat the whole argument and proceed to the next discontinuous event (if any).

Proposition 3
Assume the function f in the definition of $L(q)$ is bounded. Under assumptions (A1) - (A6), $L(q)$ is a.s. continuous in q throughout Θ.

The idea of the proof is to take a sequence in Θ converging to q and to show that the performance measures also converge. Using (4.2) and Proposition 2 we can do this on a term-by-term basis.

The next proposition provides the unbiasedness of the IPA derivative. The proof uses the continuity of $L(q)$ and applies the dominated convergence theorem to interchange the derivative and the expectation. In order to do this an additional regularity condition is needed. We give Glasserman's [3]:

(A7) There is a constant $B>0$ such that for all $q\in\Theta$, all $a\in V$, and all k,
$$\left|\frac{dX_q(a,k)}{dq}\right| \le B\left(X_q(a,k)+1\right).$$

Proposition 4
Assume the function f in the definition of $L(q)$ is bounded. Under assumptions (A1) - (A7), if $E\left[\sup_{q\in\Theta} t_f\right] < \infty$, then (4.1) holds on Θ.

To close this section, we mention an issue regarding the commuting condition, (A6), and the random number generator used to drive the simulation. Although it is not necessarily a standard procedure for implementing event graphs, a prerequisite for two events *a* and *b* not to interact is that they are assigned distinct sequences of random numbers. For example, consider the event graph fragment shown in Figure 14.

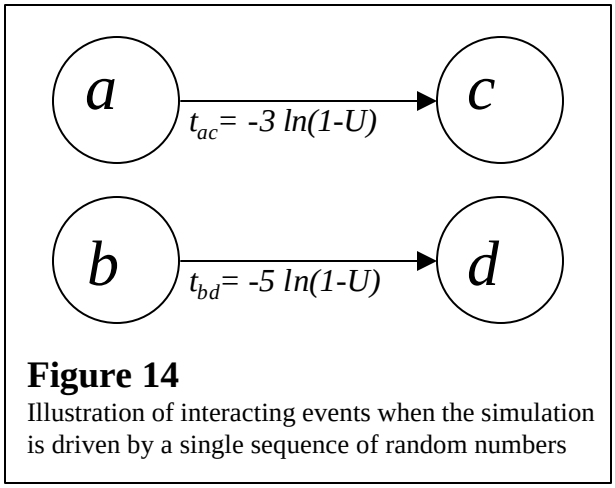

Figure 14
Illustration of interacting events when the simulation is driven by a single sequence of random numbers

Here *U* is a function that returns a random number, and edge delay times t_{ac} and t_{bd} are exponential random variables. It is apparent that *a* and *b* must interact if they use a common sequence of random numbers. The use of a distinct random number streams for different events is a typical assumption of the generic GSMP framework; see, for example, [3].

5.0 Conditional IPA and a Resource Breakdown Example

In this section we look at a more complicated example, a single server queue with service interruptions, and see what can be done when the commuting condition (A6) is violated. One approach, discussed by Fu and Hu in [1], is to use conditional IPA. We enhance the structure and notation developed in Section 3 so that results from [1] may be applied to event graphs. We also discuss some difficulties with regard to implementing conditional IPA for event graphs.

Figure 15 shows an event graph for a single server queue with service interruptions. The time between breakdowns is denoted *T*, an exponential random variable with mean *q*. When a breakdown occurs, the job being serviced is discarded. Breakdowns may occur when no job is being serviced (e.g. a photocopy machine that is always left on, even when no copies are being made).

Figure 15 also shows the (somewhat simplified) automated IPA calculations from Section 3.2 with *L(q)* being the expected waiting time. It is clear that for a long enough repair time, the derivative of the expected average waiting time with respect to *q* should be negative; increasing the interval between breakdowns reduces congestion in the system, decreasing wait times. We see from the event graph that the IPA sample path derivative will always be non-negative. In other words, the IPA estimate has the wrong sign altogether.

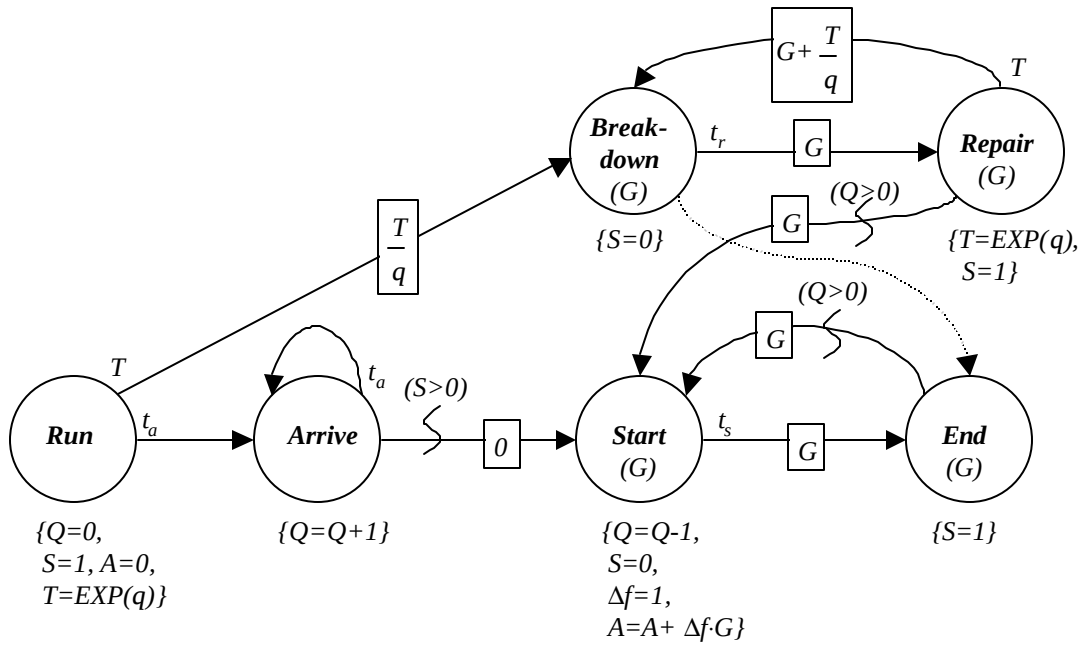

Figure 15
IPA implementation for the Resource Breakdown model

The key issue is that the commuting condition (A6) is violated; for example, the *Breakdown* and *Arrive* events, which are both in the set *H*, interact. A job that arrives to an empty system just before a breakdown will have no waiting time since interrupted services are discarded. A job that arrives just after the breakdown will have to wait the length of the repair time. Therefore a change in the order of a *Breakdown* and *Arrive* event due to an increase in q causes the overall waiting time to jump downward by the length of the repair time.

There is also a secondary interaction between *Breakdown* and *Arrive* that occurs even when the repair time is zero. For example, suppose two jobs arrive in rapid succession to an empty system. If the first job arrives just before a *Breakdown*, the second job will not have to wait for service. If the first job arrives just after the *Breakdown*, the second job may have to wait for the first job to finish service.

A similar interaction occurs between *Breakdown* and *End*. If a job finishes service while the queue is not empty, the server begins immediately on the next job. If a *Breakdown* follows, this second job is discarded. However if the *Breakdown* occurs prior to the first job's service completion, the second job will have to wait the length of the repair time. A graph of the function $L(q,w)$ for a hypothetical sample path is shown in Figure 16. The slope of the function is positive, except at points where events change order.

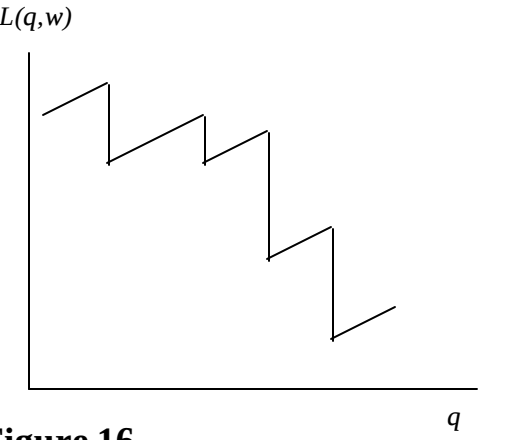

Figure 16
L(q,w) for a hypothetical sample path of the Resource Breakdown model

Examining the event graph in Figure 15 further, we see that the set *H* consists of *Arrive, End, Breakdown,* and *Repair.* The *Repair* event cannot occur simultaneously with *End* or *Breakdown,* and following the discussion of Figure 13 in Section 4.0, it is apparent that *Arrive* does not interact with *End* or *Repair* so long as *Start* has higher priority than *Arrive.* The conclusion is that *Breakdown/Arrive* and *Breakdown/End* are the only pairs of events that significantly interact.

Having identified the events that interact, the next step might be to implement a version of conditional IPA to obtain an unbiased derivative estimator. The idea is to add a term to (3.5) that represents the derivative of *L(q)* conditioned on two events in the sample path changing order. This notion is thoroughly discussed by Fu and Hu [1, pp. 85-92]. The purpose of the following paragraphs will be to enhance the structure and notation developed in Section 3, so that the analysis in [1] may be applied directly.

It is clear that if a perturbation in q causes the sample path to change, this is due to a change in the execution of two meta-events. Adapting the notation from [1], define $A_k(Dq)$ to be the subset of sample paths for which perturbing q by an amount Dq leaves the sequence of meta-events unchanged up to and including the k^{th} meta-event. Define $B_k(Dq)$ to be the subset of sample paths for which perturbing q by an amount Dq causes the sequence of meta-events to change, the first change occurring at the k^{th} meta-event.

Suppose our simulation experiment terminates at the n^{th} meta-event. We can partition the set of sample paths using $A_k(Dq)$ and $B_k(Dq)$ and write:

$$\frac{dE[L(q)]}{dq} = \lim_{\Delta q \to 0} \frac{E[L(q+\Delta q)1(A_n)] - E[L(q)1(A_n)]}{\Delta q} + \sum_{k=1}^{n} \lim_{\Delta q \to 0} \frac{E[L(q+\Delta q)1(B_k)] - E[L(q)1(B_k)]}{\Delta q} \qquad \textbf{(5.1)}$$

which is Fu and Hu's equation (3.17). An unbiased estimator for the first term in this equation is the IPA derivative given by (3.5). The second term is a derivative conditioned on a change in the k^{th} meta-event.

The approach for the second term is as follows. We have conditioned on the event B_k, a change in the k^{th} meta-event. Now focus specifically on which meta-event will be interchanged with $\bar{a}_k$. Define H_k to be the set of events on the FEL just after $\bar{t}_{k-1}$, less the lead event of $\bar{a}_k$. These are the events that could possibly be exchanged with $\bar{a}_k$ by a perturbation in q. The events in H_k are of types contained in H since they were scheduled with non-zero delay times.

Consider an arbitrary event a in H_k. The idea is to hold everything else in the simulation replication fixed, and to consider the possibility of a changing places with the lead event of $\bar{a}_k$. For this purpose (again adapting the notation from [1]), define a *characterization*, $z_k(a)$, to be the set of random numbers used by the simulation up to the n^{th} (last) meta-event, less the random numbers used to determine $X(a,\cdot)$, the clock sample used to schedule event a. Together $z_k(a)$ and $X(a,\cdot)$ determine the entire sample path. If we condition on $z_k(a)$ but not on $X(a,\cdot)$, the meta-event $\bar{a}_k$ is still random. However we have limited the candidates to the closure of a and the original $\bar{a}_k$.

This approach, conditioning in turn on the characterization $z_k(a)$ for each event a in H_k, allows Fu and Hu to isolate the effect of exchanging $\bar{a}_k$ with the particular event a. Adapting this analysis to meta-events, we have expression (5.2), which holds under some (mild) additional conditions:

$$\begin{aligned}\frac{dE[L(q)]}{dq} &= E[\text{equation } (3.5)] \\ &+ E\left[\sum_{k=1}^{n}\sum_{a\in H_k}\left\{E\left[L(q)\middle|z_k(a), X_k(a;q)=x_k^-(a)\right]\right.\right. \\ &\qquad \left.\left. - E\left[L(q)\middle|z_k(a), X_k(a;q)=x_k^+(a)\right]\right\} \\ &\qquad \left.\times \frac{\partial F_a(x;q)}{\partial x}\bigg|_{x=x_k(a)} \times\left\{\frac{dx_k(a)}{dq} + \frac{\partial F_a(x;q)/\partial q}{\partial F_a(x;q)/\partial x}\bigg|_{x=x_k(a)}\right\}^{+}\right]\end{aligned} \qquad \textbf{(5.2)}$$

where for $a \hat{I} H_k$,

$x_k(a)$ is the age of event a at time $\bar{t}_k$;
$X_k(a;q)$ is the clock sample used to schedule a;
$F_a(x;q)$ is the cdf of $X_k(a;q)$;
$x_k^-(a) = \lim_{e\to 0}[x_k(a) - e], e > 0$;
$x_k^+(a) = \lim_{e\to 0}[x_k(a) + e], e > 0$.

This is an adaptation of equation (3.29) in [1].

The double sum in (5.2) accumulates, for each meta-event k, the expected effect of exchanging $\bar{a}_k$ with each event $a \hat{I} H_k$. The expression

$$E\left[L(q)\middle|z_k(a), X_k(a;q) = x_k^-(a)\right] - E\left[L(q)\middle|z_k(a), X_k(a;q) = x_k^+(a)\right] \qquad \textbf{(5.3)}$$

is the change in the performance measure when $\bar{a}_k$ and a are exchanged. To see this, remember that together $z_k(a)$ and $X_k(a;q)$ determine the sample path, so setting $X_k(a;q)$ to $x_k(a)$ requires that a and $\bar{a}_k$ occur at the same time. In the first expectation a is executed before $\bar{a}_k$; in the second term, $\bar{a}_k$ is executed first. Thus (5.3) is the difference in the performance measure when the two events are interchanged. We can think of the second factor inside the double sum,

$$\left.\frac{\partial F_a(x;q)}{\partial x}\right|_{x=x_k(a)} \times \left\{\frac{dx_k(a)}{dq} + \left.\frac{\partial F_a(x;q)/\partial q}{\partial F_a(x;q)/\partial x}\right|_{x=x_k(a)}\right\}^+, \qquad \textbf{(5.4)}$$

roughly speaking as the rate at which the two events are interchanged given a change in parameter q.

The expression on the right side of (5.2), taken without the expectation, is an unbiased estimator for $\frac{dE[L(q)]}{dq}$. However it is difficult to develop an automatic implementation paralleling the discussion of Section 3.2. The first issue arises with regard to expression (5.3); in general this expression is difficult to evaluate. For example, consider the resource breakdown model described earlier. As we have seen, the interchange of a *Breakdown* event with an *Arrive* or *End* event can have a significant effect on the performance measure. (For some particular problems, additional analytical work can produce a version of (5.3) that is easier to evaluate. See, for example, [1].)

In principle we can evaluate (5.3) with an additional pair of simulation replications that are computed in parallel to the original replication. Fu and Hu refer to this method as "insertion" [2]. Event graphs can be modified to produce simultaneous replications by adding a parameter to each event node representing the index of the replication. (See, for example, Chapter 12 of [8].) For each event type e in H, the event graph would be modified to initiate additional replications for events interacting with e. In theory one could work out a procedure to create the necessary modifications for an arbitrary event graph, however in practice it is easier to take advantage of the structure of a particular event graph.

A secondary issue with regard to the implementation of (5.2) highlights a distinction between event graphs and GSMPs. Since the inner sum in the second term is over H_k, at time $\bar{t}_k$ we must be able to determine which events are on the FEL. This is not a problem for the generic GSMP formulation given in Section 2.1; only one event of each type may be scheduled at a time, and we can determine which events are scheduled by

knowing the values of the state variables. In an event graph we might determine which events are on the FEL by expanding the set of state variables to include, for each event $a \hat{I} V$, the variable c_a which tracks the number of instances of a on the FEL. However we must still store the time at which each instance of a was scheduled, and in principle the FEL may contain an arbitrarily large number of instances of a. In this case our event graph implementation must allow for a dynamic data structure such as a linked list.

6.0 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. Furthermore, we have taken advantage of the pictorial nature of event graphs to provide necessary conditions for unbiasedness.

ACKNOWLEDGMENTS

The National Science Foundation under grant DMI-9713549 and the Semiconductor Research Corporation under contract 97-FJ-490 supported some of the work reported in this article.

REFERENCES

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

[2] Fu, M., Hu, J., *Extensions and Generalizations of Smoothed Perturbation Analysis in a Generalized Semi-Markov Process Framework,* IEEE Transactions on Automatic Control, Vol. 37, No. 10, October 1992.

[3] Glasserman, P., *Gradient Estimation Via Perturbation Analysis*, Kluwer, Boston, 1991.

[4] Haas, P.J, Shedler, G.S., *Stochastic petri nets: Modeling power and limit theorems,* Probability Eng. Inform. Sci., vol. 5, 1991, pp. 477-498.

[5] Iglehart, D. L. and Shedler, G. S., *Simulation of Non-Markovian Systems,* IBM Journal of Research and Development, Vol. 27, No. 5, pp. 472-479.

[6] Jacobson, S., Yücesan, E., *On the Complexity of Verifying Structural Properties of Discrete Event Simulation Models,* Operations Research, Vol. 47, No. 3, May-June 1999.

[7] Schruben, L., *Simulation Modeling with Event Graphs,* Communications of the ACM, Vol. 26, No. 11, November 1983, 957-963.

[8] Schruben, L., *Graphical Simulation Modeling and Analysis*, Boyd & Fraser Publishing Company, Danvers, Mass., 1995.

[9] Schruben, L., Savage, E., *Visualizing Generalized Semi-Markov Processes*, Proceedings of the 1996 Winter Simulation Conference, ed. Charnes, J., Morrice, D., Brunner, D., Swain, J., 1996.

[10] Schruben, L., Yucesan, E., *Structural and Behavioral Equivalence of Simulation Models*, ACM Transactions on Modeling and Computer Simulation, vol.2, no.1 p. 82-103, January 1992.

[11] Som, T., Sargent, R., *A Formal Development of Event Graphs as an Aid to Structured and Efficient Simulation Programs*, ORSA Journal on Computing, Vol. 1, No. 2, Spring 1989, 107-125.

[12] Suri, R., *Infinitesimal Perturbation Analysis for General Discrete Event Systems*, Journal of the Association for Computing Machinery, Vol. 34, No. 3, July 1987, 686-717.

[13] Xie, X., *Perturbation Analysi of Stochastic Petri Nets*, IEEE Transactions on Automatic Control, Vol. 43, No. 1, January 1998.