Proceedings of the 2003 Winter Simulation Conference
S. Chick, P. J. Sánchez, D. Ferrin, and D. J. Morrice, eds.

ADVANCED EVENT SCHEDULING METHODOLOGY

Lee W. Schruben
Theresa M. Roeder
Wai Kin Chan

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

Paul Hyden

Department of Mathematical Sciences
O-309 Martin Hall
Clemson University
Clemson, SC 29634-0975, U.S.A.

Mike Freimer

Department of Management Science and
Information Systems
509N Business Administration Bldg.
The Pennsylvania State University
University Park, PA 16802, U.S.A.

ABSTRACT

Simulation Event Graphs (SEGs) are a graphical representation one of the three major simulation world views, event scheduling. This paper describes four advanced modeling techniques that allow the simulation practitioner to gather a great deal of information at relatively little development and/or processing effort beyond that of developing the simulation model.

1 INTRODUCTION

One of the three main approaches to discrete event simulation modeling is the event scheduling world view, see (Derrick, *et al.* 2000). One key advantage to this world view is its speed in executing models of congested systems (Schruben and Roeder 2003). In addition, Simulation Event Graphs (SEGs), a type of event scheduling simulation, are able to compactly represent large models, and can be analyzed using optimization and graph theory.

This paper describes advanced modeling techniques that are available in the event scheduling world view. Section 2 formally defines SEGs to aid the development of future sections. Section 3 introduces the ideas of simultaneous replications and time dilation for simulation optimization; Section 4 illustrates infinitesimal perturbation analysis (IPA) of SEGs; Section 5 reports recent results in using math programming together with SEGs; and Section 6 describes ongoing work in estimating rare event behavior.

2 SIMULATION GRAPHS

This section will give a formal definition of SEGs. The definition will be illustrated in a simple example.

2.1 Definition

Simulation Event Graphs model the dynamics of discrete event systems. We will use notation from (Yücesan and Schruben 1992). Accordingly, for a graph G, define:

$V(G)$	set of event vertices in G
$E_s(G)$	set of scheduling edges in G
$E_c(G)$	set of canceling edges in G
Ψ_G	incidence function on G, associating an ordered pair of vertices with each edge in G

The pairs of vertices in Ψ_G need not be distinct. A simulation graph is defined as the ordered quadruple $G = (V(G), E_s(G), E_c(G), \Psi_G)$. The vertices $V(G)$ represent the *events* that occur, causing a change in the system state. The (directed) edges $E_s(G)$ and $E_c(G)$ indicate relationships between the events, as specified by Ψ_G.

Let $\mathbb{R}^+$ be the set of non-negative real numbers, and let *STATES* denote the set of possible states for the underlying model. Then we additionally define the following data indexed by sets in G:

- set of state transition functions associated with vertex v: $\mathcal{F} = \{f_v\text{: STATES} \rightarrow \text{STATES} \mid v \in V(G)\}$

- set of *edge conditions* associated with edge e: $\mathcal{C} = \{c_e: STATES \rightarrow \{0,1\} \mid e \in E_s(G) \cup E_c(G)\}$
- set of *edge delay times* associated with edge e: $\mathcal{T} = \{t_e: STATES \rightarrow \mathfrak{R}^+ \mid e \in E_s(G) \cup E_c(G)\}$
- set of *event execution priorities* associated with edge e: $\Gamma = \{\gamma_e: STATES \rightarrow \mathfrak{R}^+ \mid e \in E_s(G)\}$
- set of *event parameters* associated with vertex v: $\mathcal{P} = \{p_v: STATES \rightarrow \mathfrak{R}^+ \mid v \in V(G)\}$
- set of *edge attributes* associated with edge e: $\mathcal{A} = \{a_e: STATES \rightarrow \mathfrak{R}^+ \mid e \in E_s(G) \cup E_c(G)\}$

A simulation graph model is the ordered seven-tuple $\mathcal{S} = (\mathcal{F}, \mathcal{C}, \mathcal{T}, \Gamma, \mathcal{P}, \mathcal{A}, G)$. The first six sets identify entities in the model, while G creates a meaningful model by specifying the relationships among indices for elements in $\mathcal{F}, \mathcal{C}, \mathcal{T}, \Gamma, \mathcal{P}$, and $\mathcal{A}$.

2.2 Example

Consider a simple G/G/s queueing system. The state of this system is the number of jobs waiting to be served, Q, and the number of available servers, R. The set of all possible *STATES* for this model is $\{(Q,R): Q = 0, 1, 2,\ldots, R = 0, 1,\ldots,s\}$. Additional state variables are $W[i]$, the time the ith customer spends in the system; *ID*, the customer identification number; *IN*, the customer ID of the customer currently in service (the last to have started or completed service); *NEXT*, the customer ID of the next customer to begin service; and *CLK*, the simulation clock time. The (random) customer interarrival times are t_a, and service times are t_s.

The set of event vertices $V(G)$ consists of the events *ENTER*, *START*, and *LEAVE*. $E_c(G) = \varnothing$; the scheduling edges $E_s(G)$ are shown in Table 1.

Table 1: Edges for the SEG of G/G/s Queue

Edge	Vertex pair
1	*ENTER – ENTER*
2	*ENTER – START*
3	*START– LEAVE*
4	*LEAVE – START*

The state changes $\mathcal{F}$ and event parameters $\mathcal{P}$ for each event are given in Table 2.

The edge conditions $\mathcal{C}$, time delays $\mathcal{T}$, event priorities Γ (associated with the edge that schedules/cancels the event), and edge attributes $\mathcal{A}$ are given in Table 3.

Initially, R = s, Q = 0, CLK = 0, and the *ENTER* event starts the simulation run.

Table 2: State Changes and Parameters for Events in the SEG of the G/G/s Queue

Event	*State changes*	*Event parameters*
ENTER	Q=Q+1, ID=ID+1, W[ID]=CLK	
START	Q=Q-1, R=R-1, NEXT=NEXT+1	IN
LEAVE	R=R+1, W[IN]=W[IN]-CLK	IN

Table 3: Data Associated with Edges for the SEG of the G/G/s Queue

Edge	*Condition*	*Time delay*	*Priority*	*Attributes*
1		t_a	2	
2	R > 0		1	ID
3		t_s	2	IN
4	Q > 0		1	NEXT

Throughout this paper, the term "model" will be used to refer to one complete specification from a design region or search space. Thus, a system where s = 3 is a different model than one where s = 4.

3 SIMULTANEOUS REPLICATIONS AND TIME DILATION

The ideas of simultaneous replications and time dilation were formally proposed in (Schruben 1997). Simultaneous replications allow multiple (parameterized) runs to be done during the same simulation execution. This is likely to cause the future events list (FEL) to become extremely large; however, it is actually possible to take advantage of the large list using time dilation.

Fundamentally, time dilation increases the "time scale" for scenarios that are performing poorly relative to the others. This is done by penalizing less promising scenarios. While the simulation does not completely stop collecting data for these scenarios, it does not spend as much time on them but tries to improve estimates for more promising configurations. Experimental results have shown the simultaneous replications and time dilation can quickly find correct solutions (for problems where the answer is known) with relatively little expended computational effort.

In (Schruben 1997), time dilation does just that, it changes the time scale. Several M/M/1 queues are simultaneously simulated with different release rates to determine the 4X capacity/release rate. That is, the job arrival (release) rate that results in the average cycle time is four times the raw job processing time. If $\overline{W_i}$ is the average job delay for release rate i and P is an exponential scaling factor, the time scale for the system with release rate i was multiplied by $\left(\overline{W}_i - 4\right)^P$. (For example, *P=2* if $\overline{W}_i$<4 and *P=4* otherwise.) The releases for less promising rates are scheduled for later points in time, and, in effect, less simulation time is spent on them. The results show impressively that the vast majority of the events executed during the run are associated with the "correct" release rate.

The ideas proposed in (Schruben 1997) are developed and refined in (Hyden and Schruben 1999, 2000; Hyden 2003). An important difference is a reinterpretation of the concept of time. Rather than changing the time scale of the concurrent models, units of time are seen as number of events executed. Scenarios with more promising results are given more attention than those with less promising results. The scenario to devote the next segment of CPU time can be determined probabilistically, where each situation is assigned a probability based on its relative performance.

The example illustrated in (Hyden and Schruben 2000) is that of a job shop with five server types. One additional server is available, and can be added to any of the five groups. The five possible scenarios (the additional server being assigned to each of the groups) are run simultaneously. The ultimate objective is to find the setup with the smallest expected job time in system. Since it cannot be guaranteed that every run will find the optimal setup, the "working" objective is to maximize the probability of selecting the correct system.

In this example, the quality of a setup is based on a distance measure D_i between model *i* and its best competitor. Additionally, the sample variance V_i of the expected performance measure is calculated. Each model *i* is assigned a score of V_i / D_i^2, and is selected for execution with probability V_i / D_i^2 over the sum of all of the scores. This favors models with smaller distance measures D_i and also those with greater variance (to try to reduce the variance).

The simulation engine executes one event from the selected model, and then reevaluates which model it should choose next. Probabilities p_i are recalculated every 100 events.

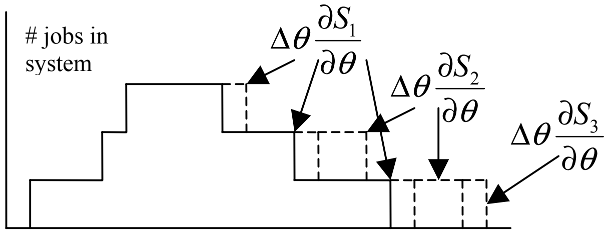
The simultaneous replications were run with and without common random numbers (CRN), and it is shown that the CRNs improve the probability of selecting the correct solution substantially.

There are many opportunities for changes and enrichments to the basic structure introduced here. Certainly estimates of V_i can be improved upon by taking into account serial dependency. Scores could be modified to weight recent data more heavily than past data. Rather than executing one event and reselecting models, the engine could execute a certain number of events from the model (perhaps the number could be score-dependent) before reevaluating.

4 INFINITESIMAL PERTURBATION ANALYSIS

Infinitesimal perturbation analysis (IPA) is a technique that has been studied for many years. It allows the simulation practitioner to estimate response gradients (with respect to several parameters) for the system modeled with a single run. In contrast, finite differencing requires two runs to find the response derivative for a single parameter. While most IPA results have been developed for Generalized Semi-Markov Processes (GSMPs), see for example (Glasserman 1991), in this paper, we will focus on a procedure developed in (Freimer 2001) to easily perform IPA estimation using SEGs.

The following discussion will use notation from (Freimer and Schruben 2001a, 2001b). Consider an M/M/1 queue with mean service time θ. Let $S_i(\omega,\theta)$ be a realization of job *i*'s service time, using random seed ω. If θ were increased by $\Delta\theta$, this service time would be increased by $\Delta\theta \frac{\partial S_i(\omega,\theta)}{\partial\theta} + o(\Delta\theta)$. Figure 1 shows a sample path for the number of jobs in the system (waiting and in service) for the M/M/1 queue. The number increases when a job arrives and decreases when a job finishes service. The area under the solid line is the total delay experienced by the jobs in the unperturbed system. The area under the dashed lines is the additional delay experienced by jobs because of the increase in θ. It is important to note that the additional delays are cumulative in a busy period: The third job in a busy period will have to wait for the additional service times jobs 1 and 2 experienced, and will also be delayed by its own increase in service time.



# jobs in system
$\Delta\theta\frac{\partial S_1}{\partial\theta}$
$\Delta\theta\frac{\partial S_2}{\partial\theta}$
$\Delta\theta\frac{\partial S_3}{\partial\theta}$

Figure 1: Sample Busy Period for M/M/1 Queue

If the purpose of the study is to find the derivative of the average waiting time $W(\omega,\theta)$ (which is the average delay minus the average service time), it can be expressed as the sum of the derivatives of the S_i. Let *N* be the total number of jobs served, *M* the number of busy periods, and

k_m the index of the first job in busy period m. Then the sample path derivative of the waiting time $W(\omega,\theta)$ is

$$\frac{dW(\omega,\theta)}{d\theta} = \frac{1}{N}\sum_{m=1}^{M}\sum_{j=k_m}^{k_{m+1}-1}\left(k_{m+1}-1-j\right)\frac{\partial S_j(\omega,\theta)}{\partial\theta} \quad (1)$$

This expression does not depend on $\Delta\theta$, so we need only track the service time derivatives. In the case of exponential service times, a service time $S(\omega,\theta)$ is generated using a uniform random number $U(\omega)$: $S(\omega,\theta) = F_S^{-1}(U(\omega),\theta) = -\theta\ln(1-U(\omega))$. Its derivative is $\frac{\partial S(\omega,\theta)}{\partial\theta} = -\ln(1-U(\omega)) = \frac{1}{\theta}S(\omega,\theta)$.

The derivative of $W(\omega,\theta)$ is a sample path derivative. Conditions for unbiasedness of the derivative of the expected waiting time are outlined in (Freimer and Schruben 2001b).

While the equation in (1) is correct, it requires knowing the length of the busy period, information that is not known during a simulation run (until the busy period is over). Nonetheless, gathering sample path service derivatives during a run is fairly straightforward with SEGs.

The average time in system of a job can be calculated as the integral over time of the number of jobs in the system (divided by N), which is equivalent to the area under the curve in Figure 1. However, we can also note that this is the sum of the product of the number of jobs in the system and the length of time this state remains unchanged. If we let $f(s_i)$ be the number of jobs in the system after the ith state change has occurred, τ_i be the time of this ith state change, and $N(t_f)$ be the number of system changes that have occurred by the simulation finish time t_f, the average time in system can be expressed as $W(\omega,\theta) = \frac{1}{N}\mathop{}_{i=0}^{N(t_f)-1} f(s_i)(\tau_{i+1}-\tau_i)$. Further defining Δf_{si} as $f(s_{i-1})$-$f(s_i)$ if $i < N(t_f)$ and $f(s_{i-1})$ if $i < N(t_f)$, we can express the derivative of $W(\omega,\theta)$ (after some algebra) as $\frac{dW(\omega,\theta)}{d\theta} = \frac{1}{N}\mathop{}_{i=1}^{N(t_f)}\Delta f_{s_i}\frac{d\tau_i}{d\theta}$. The change in τ_i can be expressed as the changes in the edge delay times during the busy period that led up to τ_i, as seen in Figure 1.

To finally implement the estimator in the SEG, we define the following two variables: A is the accumulator, which stores the total changes in waiting time accrued thus far. G is used to pass the delay time derivatives. Both A and G are initialized to zero at the beginning of the run. Each edge e with delay t_e will have the additional attribute of $G+dt_e/d\theta$. Each event s will have two additional state changes: $f=\Delta f_s$ (as appropriate), and $A=A+fG$.

For our M/M/1 queue, Δf_{si} is -1 if s_i is an *ENTER* event, 1 if it is a *START* event, and 0 otherwise. (Note that here, we are calculating the average *waiting* time directly, since it is the average time in system minus the average service time. The service time can be subtracted from the calculation immediately, giving us the average waiting time.) Since we are only varying the service rate parameter, the derivative of the other edge delay (interarrival time) with respect to this is zero.

We need now only add the following state changes and edge attributes to the model defined in Section 2.2: The *START* event sets f=1 and $A=A+G$; and the edge from *START* to *FINISH* sends $G+t_s/\theta$. All other edges simply pass the current value of G.

5 MATH PROGRAMMING OF SYSTEM TRAJECTORIES

An advantage of simulation event graph models is that any information about the system being studied is available. A disadvantage is that the computational effort to obtain the information may be prohibitive, and great care must be taken to get accurate performance estimates.

This section describes ongoing research that links simulation and math programming models. It builds on work presented in (Schruben 2000). A big advantage of math programming models, specifically linear programs (LPs), is that there are large bodies of research on efficient solution procedures, and on sensitivity analysis. (Schruben 2000) shows that simulation models of simple queueing systems (G/G/1 and G/G/2 queues, and multiserver tandem queues) can be formulated as linear programs. The duals of the linear programs are network graphs; these graphs can be solved very quickly. Their solutions are the trajectories of the corresponding SEG. In addition, they give insights into the sensitivity of the solution to model parameters (e.g. interarrival and service times).

Consider the example from Section 2.2 where s = 1. For the time being, we will ignore the job waiting times (variables W, ID, IN, and $NEXT$). The linear program is given below in (2). The objective function is to minimize the sum of the finish times. A_i is the arrival time of the ith job, and F_i is its finish time. The service duration of the ith job is ts_i. The total number of jobs processed is N. The associated dual variable names are given in parentheses next to the constraints.

$$\begin{array}{llll} \min & \sum_{i=1}^{N} F_i & & \\ s.t. & F_i \geq A_i + ts_i & (U_i) & i=1,\dots,N \\ & -F_{i-1} + F_i \geq ts_i & (V_i) & i=2,\dots,N \end{array} \quad (2)$$

The dual LP is given in (3). Constraint i is associated with the i^{th} finish time. The constraints are totally unimodular, and the solution can be found extremely quickly. In addition, the solution values will be integer. The problem data (interarrival and service times) only appear in the objective function. The surprising (and pleasing) result is that the dual variables tell the number of jobs in the busy period (U_i) and the remaining number of jobs in the busy period (V_i). Thus, given the input data, we can solve a LP (which has taken as few as zero basis pivots) and find the busy periods, the lengths of the busy periods, and the job finish times. In addition, we can use sensitivity analysis information to determine what effect changes in the data (interarrival and service times) will have on the system behavior.

$$\begin{aligned} \max\ & \sum_{i=1}^{N}\left(A_i + ts_i\right)U_i + \sum_{i=2}^{N} ts_i V_i \\ s.t.\ & U_1 - V_2 \le 1 \\ & U_i + V_i - V_{i+1} \le 1 \quad i=2,\ldots,N-1 \\ & U_N + V_N \le 1 \end{aligned} \tag{3}$$

Figure 2 shows the dual graph for N=3. The dual variables can be interpreted as the amount of flow across their arcs, and the shaded values are the revenues earned by a unit flow across the arc. Each of the F_i nodes has a "demand" of one unit, and we are trying to create a "max flow" from A_0 to F_i, i = 1, 2, 3. For G/G/1 queues, this is known as the lot-sizing problem. See (Chan and Schruben 2003) for more details on dual formulations and their interpretations.

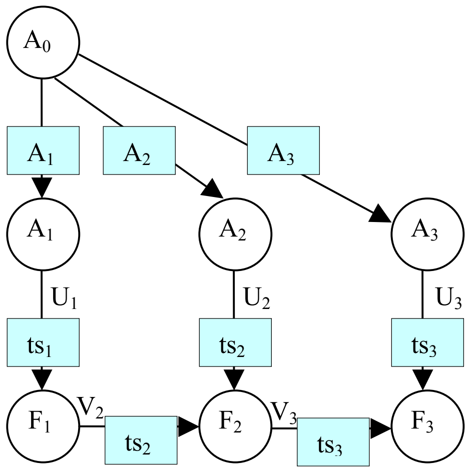


Figure 2: Dual Graph for G/G/1 Queue with Three Jobs

In (Chan and Schruben 2003) show further results for tandem queues. They use the LP formulations to show reversibility of queueing systems with blocking. Although (Schruben 2000) uses integer (assignment) variables for the formulations of G/G/s queues, we have been able to formulate them as LPs, albeit with a large number of constraints.

6 RARE EVENT ESTIMATION

The G/G/s model described in Section 2.2 tracks the waiting times for individual customers. When systems get large, this can become cumbersome because of the amount of memory and processing time required. In (Schruben and Roeder 2003), the authors describe the drawbacks of *job-driven* simulations, where each job is tracked. They suggest that *resource-driven* simulations may be more appropriate for most purposes; here, only counts of jobs and available resources are kept. This makes modeling aspects of systems and obtaining certain output statistics such as waiting time distributions more difficult.

This section describes ongoing research in estimating waiting time distributions without tracking individual jobs. We will restrict ourselves to FIFO G/G/1 systems here. Before explaining how to estimate delay times without maintaining records of each job, we will describe an "intermediate" step for estimating the probability a job has to wait less than some set time delay L.

To do so, we will supplement the model described in Section 2.2 by an additional event *DELAY*. It will be scheduled unconditionally by the *ENTER* event, and will occur L time units after a job arrives. Its state changes will consist of incrementing a delay counter D by one. If, when this *DELAY* event occurs, there have been more *START* events than *DELAY* events, we know that the job began service before it was delayed L time units. To capture this, we increment a service counter ST by one every time a *START* event occurs. The *DELAY* event will have additional state changes counting the number of jobs that have begun service before their "time was up," and updating the estimate of the probability of waiting less than L: The counter W is incremented by one if ($D \le ST$) is true, and the probability *PROB* is updated to W/ST. The value of *PROB* at the end of the simulation run will be our estimate of the probability that $W<L$.

This approach will give an accurate estimate of the probability, and works for FIFO G/G/s queues with s>1 as well. Its disadvantage is that, though we are *not* directly tracking each job, we *are* maintaining an event (*DELAY*) for each job on the FEL. Since the *DELAY* event does not schedule other events, we can dispense with it and instead just use an array to track the times the job would have been delayed L time units. This saves storage space on the order of magnitude of the number of jobs in the system, O(N). The order of the approach we will describe next is independent of the number of jobs.

The "bin" approach divides the simulation time line into equally-sized bins. When a job arrives at time t, we update the number of jobs that will have been delayed L

time units for the bin that contains time *t*+*L*, and all subsequent bins. Figure 3 shows the step functions for the *DELAY* and *START* events (dashed and normal lines, respectively), and also shows the resulting "bin" step function for bin size *b* (bold line). The *ENTER* step function is not shown. It is the same as the *DELAY* function, shifted to the left *L* units.

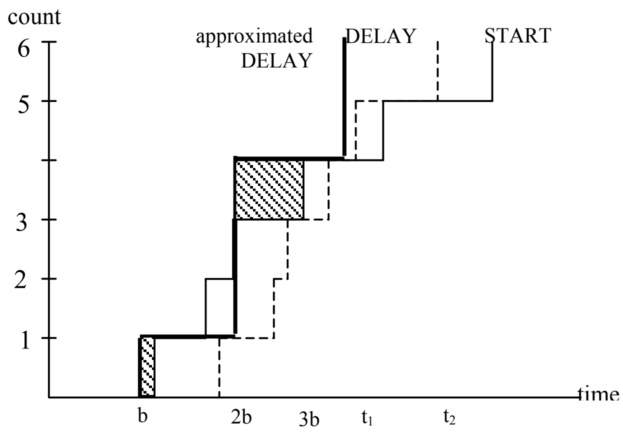


Figure 3: Step Functions for G/G/1 Queue

The original approach compares the height of the *DELAY* and *START* curves at time *t*. If the *DELAY* curve is above the *START* curve, the job has waited too long. The "bin" approach compares the height of the *START* curve to the height of the *BIN* curve. If the *BIN* curve is above the *START* curve, we will classify the job as having waited longer than *L*. We should note that this will lead to the misclassification of some jobs. In Figure 3, specifically, jobs 1 and 4 will be misclassified, as indicated by the shaded areas. To count the number of jobs misclassified, we can count the shaded rectangles in the graph. These are the areas where the *BIN* curve occurs before the *START* curve, though the *DELAY* curve itself happens after the *START* curve. The size of the rectangle does not give any indication about the magnitude of the error in the misclassification. It does show how big a tolerance there is for the actual *START* time – if the *START* occurs anywhere in that region, the job will be misclassified. Overall, our estimate of the probability of waiting less than *L* will be underestimated. That is, we will think our system is performing more poorly than it actually is.

The magnitude of the error clearly depends on the bin size. As the bin size approaches 0, we will move closer and closer to the complete job-driven case. The error will also depend on the parameters of the model. If the system is very lightly loaded, we will be less prone to error since the *START* event will happen much sooner than the *DELAY* event will, assuming *L* is not close to zero.

Figure 4 shows results for an M/M/1 queue with interarrival rate 2/3 and service rate 1. It shows that, as expected, the estimates of the probability get worse as the bin size increases. This is especially true for smaller delays. As the delays get larger, all estimates become more accurate.

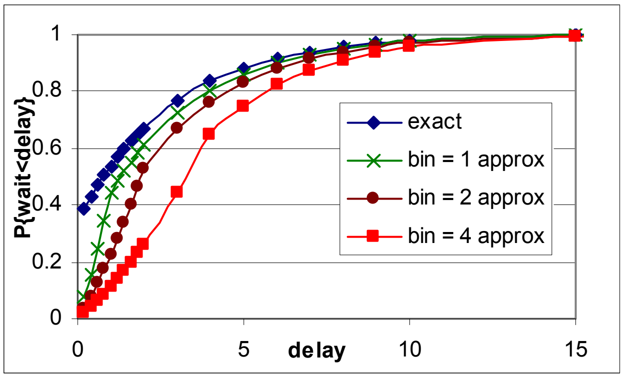


Figure 4: P{wait < delay} for M/M/1 Queue and Different Bin Sizes

For the simple M/M/1 system, the run times for the "bin" model do not result in a great speed-up compared to models that track individual jobs and return the exact waiting times. However, we are exploring the possibilities of incorporating bin approximations into large models, where the additional processing required to increment all bin counters may still be less than that of tracking individual jobs. We are also investigating the possibility of improving the approximation in the bins themselves by adding a probability that a job may actually not be delayed even though the bin counter indicates it is.

ACKNOWLEDGMENTS

Conversations with Deborah Pederson and Nirmal Govind have been extremely helpful. This work was supported in part by the Semiconductor Research Consortium and the National Science Foundation.

REFERENCES

Chan, W. K., and L. W. Schruben. 2003. Properties of Discrete Event Systems from Their Mathematical Programming Representations. *Proceedings of the 2003 Winter Simulation Conference*. ed. S. Chick, P. J. Sanchez, D. Ferrin and D. J. Morrice.

Derrick, E., O. Balci, R. Nance, and H. Shen. 2000. *Conceptual Frameworks for Discrete-Event Simulation Modeling*. Computer Science Working Paper Virginia Polytechnical Institute. Blacksburg, VA.

Freimer, M. 2001. *Integrating Data Collection and Model Analysis in Simulation*. Ph.D. Dissertation, School of Operations Research and Industrial Engineering, Cornell University. Ithaca, NY.

Freimer, M., and L. W. Schruben. 2001a. Graphical Representation of IPA Estimation. *Proceeding of the 2001*

Winter Simulation Conference. ed. B. A. Peters, J. S. Smith, D. J. Medeiros and M. W. Rohrer. Piscataway, NJ, USA: IEEE. pp.422-427.

Freimer, M., and L. W. Schruben. 2001b. *Visualizing Infinitesimal Perturbation Analysis Estimators*. Technical Report 1291, School of ORIE, Cornell University. Ithaca, NY.

Glasserman, P. 1991. *Gradient Estimation Via Perturbation Analysis*. Kluwer Academic Publishers, Norwell, MA.

Hyden, P. 2003. *Time Dilation: Decreasing Time to Decision with Discrete-Event Simulation*. Ph.D. Dissertation, School of Operations Research and Industrial Engineering, Cornell University. Ithaca, NY.

Hyden, P., and L. Schruben. 1999. Designing Simultaneous Simulation Experiments. *Proceedings of the 1999 Winter Simulation Conference*. ed. P. A. Farrington, H. Black Nembhard, D. T. Sturrock and G. W. Evans. Piscataway, NJ, USA: IEEE. pp.389-394.

Hyden, P., and L. Schruben. 2000. Improved Decision Processes through Simultaneous Simulation and Time Dilation. *Proceedings of the 2000 Winter Simulation Conference*. ed. J. A. Joines, R. R. Barton, K. Kang and P. A. Fishwick. Piscataway, NJ, USA: IEEE. pp.743-748.

Schruben, L. W. 1997. Simulation Optimization Using Simultaneous Replications and Event Time Dilation. *Proceedings of the 1997 Winter Simulation Conference*. ed. Winter Simulation Conf. Board of Directors. pp.177-180.

Schruben, L. W. 2000. Mathematical Programming Models of Discrete Event System Dynamics. *Proceedings of the 2000 Winter Simulation Conference.* ed. J. A. Joines, R. R. Barton, K. Kang and P. A. Fishwick. Piscataway, NJ, USA: IEEE. pp.381-385.

Schruben, L. W., and T. M. Roeder.2003. Fast Simulations of Large-Scale Highly Congested Systems. *Simulation: Transactions of the Society for Modeling and Simulation International*.

Yücesan, E., and L. Schruben.1992. Structural and Behavioral Equivalence of Simulation Models. *ACM Transactions on Modeling & Computer Simulation*. **2**(1): 82-103.

AUTHOR BIOGRAPHIES

LEE W. SCHRUBEN is a Professor in and Chair of the Department of Industrial Engineering and Operations Research at UC Berkeley. His email address is <schruben@ieor.berkeley.edu>.

THERESA M. ROEDER is a Ph.D. candidate in the Department of Industrial Engineering and Operations Research at UC Berkeley. Her email address is <roeder@ieor.berkeley.edu>.

WAI KIN CHAN is a Ph.D. student in the Department of Industrial Engineering and Operations Research at the University of California, Berkeley. His e-mail is <kin@ieor.berkeley.edu>.

PAUL HYDEN is an Assistant Professor in the Department of Mathematical Sciences at Clemson University. He received all of his degrees in Operations Research and Industrial Engineering from Cornell University. His email address is <hyden@clemson.edu>.

MIKE FREIMER is an Assistant Professor in the Department of Management Science and Information Systems at the Smeal College of Business at the Pennsylvania State University. His email address is <mbf10@psu.edu>.