Proceedings of the 2000 Winter Simulation Conference
J. A. Joines, R. R. Barton, K. Kang, and P. A. Fishwick, eds.

MATHEMATICAL PROGRAMMING MODELS OF DISCRETE EVENT SYSTEM DYNAMICS

Lee W. Schruben

Industrial Engineering and Operations Research
4117 Etcheverry Hall
University of California
Berkeley, CA 94720-1777, U.S.A.

ABSTRACT

Analytical models for the dynamics of some discrete event systems are introduced where the system trajectories are solutions to linear and mixed-integer programs.

1 BACKGROUND

The dynamics of continuous systems are often modeled by a set of differential equations that express the relationships between rates of *changes* in the values of system state variables. Given an initial state and boundary conditions, these equations completely specify a model of the system's dynamic behavior. When this system of differential equations is particularly simple or has some special properties, it can be solved analytically to find the system's path of motion (trajectory). However, many interesting models are too complex to solve analytically and must be simulated by numerically integrating the set of differential equations. If the system is modeled using random processes, then the simulations can be used to generate sample paths for statistical analysis.

In a somewhat analogous manner, the relationships between *changes* in the values of state variables (events) in a discrete event system can be modeled with an event graph. The vertices of the graph represent state changes and the edges of the graph represent the dynamic and logical relationships between these changes. An event graph, along with initial conditions, completely specifies the discrete event system dynamics. As is the case for continuous systems, the dynamics of most discrete event system models are complex and must be simulated. In this paper, we look at some discrete event models where the system of difference equations embedded in their event graphs can be solved analytically for the system trajectory.

Queueing systems are an important class of discrete event dynamic systems. The event graph models for some queueing systems have state trajectories that are the solutions to mathematical optimization programs. We will denote the class of simulation event graph models for which these mathematical programs are mixed integer programs as *S*. The subset, $L \subset S$, contains those event graphs whose state trajectory can be found as the solution to a linear program. If is possible to develop different event graph models that represent different levels of detail or different sets of state variables for a particular discrete event system. Some of these have related but distinct mathematical programs that specify their trajectories.

Beyond a purely academic interest, the are several potential reasons why we might want to express the dynamics of a discrete event system as a mathematical program. The mathematical programming model for the system dynamics provides constraints that must be enforced in an associated optimization problem for resource scheduling. Instead of generating scheduling constraints in the usual ad-hoc manner, these constraints might be obtained more or less methodically from the system's dynamic model. Another reason why we might be interested in an analytical model for a system's dynamics is to allow us to "run" a simulation model in *S* by solving the associated optimization problem. Alternatively, the simulation model can be executed in the usual manner to provide an optimal solution to the optimization problem. This solution might then be useful as the starting place for performing a sensitivity analysis of the system's performance to parametric and structural changes. The mathematical programs could also be used to model subsystems in a hierarchical simulation.

After reviewing event graph models, a class of models called resource-driven simulations is defined. These simulation models are in *S* and have some other advantages and disadvantages over conventional job-driven (process) simulation models. Examples obtaining analytical solutions for the trajectories of these models are presented. Finally, some possible applications are proposed for further investigation.

2 EVENT GRAPHS

A common way of modeling many queueing systems is with a state transition diagram (Ross, 1993). The vertices in this graph represent values of each system state variable.

Even a simple M/M/1 queue requires a state transition diagram with an infinite number of vertices.

A more compact graph representation for the dynamics of a queueing system (or any other discrete event system) is to represent only the *changes* in system state by the vertices. Each vertex in this graph represents one or more difference equations that specify the state changes associated with a system event. Such a graph is called an event graph (Schruben and Schruben 2000). The vertices of an event graph represent the state changes that result when a particular event occurs. These vertices are connected by directed edges that represent the relationships between the events. Labels on these directed edges specify the conditions and time delays between the occurrences of events. An edge in an event graph is shown in Figure 1.

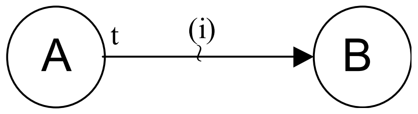

Figure 1: Element of an Event Graph

The edge in Figure 1 is interpreted as follows: *whenever event A occurs, if condition (i) is then true, event B will be scheduled to occur after a delay of t.*

The state changes associated with each event appear as vertex labels in braces. Parameter values can be passed as arguments for event vertices allowing very large systems to be modeled with small graphs. Treating these parameters as subscripts, the graph then describes an element in an array of event graphs working together to model a large system. For example: networks of queues can be modeled by specifying an event graph for only a generic node in the network; vertex parameter values distinguish each node. Any of the objects in an event graph can themselves be event graphs. For example: the delay time, t, for an edge in a factory-level model can be computed as the solution to a more detailed event graph model of the particular tool or process involved.

In this paper we will use a simple event graph model for illustration. Figure 2 is a batch processing system with R identical parallel resources.

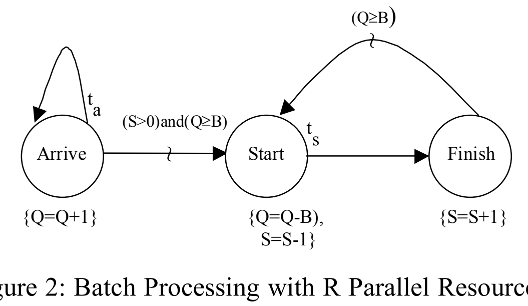

Figure 2: Batch Processing with R Parallel Resources; Q = Queue Length, B = Batch Size, S = Idle Resources, t_s = Service Times, t_a= Interarrival Times

Initially, the queue is assumed empty with all the resources idle. Thus the initial value of the line length, Q, is set to zero and the initial value for the number of idle resources, S, is set equal to R. The only event initially scheduled is the first Arrive event, at time zero. Of course, there are several equivalent event graphs for this system. In fact, special cases of this system can be modeled with only a single "Finish" event vertex.

We will use Figure 2 to illustrate how to "read" an event graph. Event graphs are read by substituting for A, B, (i), and t in the edge definitions and paraphrasing. An accurate and concise description of the system dynamics can be given with only *one sentence for each edge* in the graph. The four edges of the event graph in Figure 2 are read as the following four sentences.

> Jobs *arrive* every t_a time units. Jobs that *arrive* to find an idle resource and who complete a full batch will immediately *start* service. A resource will *finish* processing a batch of jobs t_s time units after it *start*s. When a resource *finish*s a batch, if a full batch of jobs is waiting, the resource can immediately *start* on the next batch.

3 ESOURCE-DRIVEN SIMULATIONS

The discrete event system trajectories generated by most conventional simulation languages use a numerical algorithm and modeling worldview that is sometimes referred to as "job-driven" or "process interaction". The jobs are the *active* system entities that "seize" available system resources when they need them. This approach typically requires that every step in the processing of every job in the system be explicitly represented. Records of all jobs in the system are created and maintained. The jobs move through their processing steps (often represented by a block flow diagram) seizing available system resources whenever they are needed. Job-driven models are convenient when predicting system performance and fast simulation execution speed are not as important as detailed system animation - there often is a direct mapping from simulation code to animation.

An advantage of using a job-driven event graph model over a resource-driven model is that the detailed experiences of individual jobs can be easily tracked. This is important, say, in modeling low-volume, high-mix manufacturing systems. A major disadvantage of job-driven simulations is that whenever the simulation becomes highly congested, the simulation's memory footprint becomes larger, and its execution slows or stops completely. It is easy to represent job-driven (process interaction) simulations with an event graph; however, we will look at the system from a resource viewpoint.

In the model considered in this paper, the individual transient entities (jobs) in the system are *passive*, they are "moved" or "processed" by the system's resources. We will refer to such simulation models as “resource-driven” simulations. Rather than maintaining a record of every job in the system, only integers that count the numbers of jobs of particular types at different stages of processing or in different states are necessary. The system’s state is described by the availability of resources (also integers) and these job counts. Thus, all the state variables in a resource-driven simulation are non-negative integers. The state changes for each event are difference equations that increase or decrease one or more state variables by integer amounts. We will also need a property on the edges of each vertex to derive analytical models for an event graph - all edge conditions are in the form of linear constraints on the state variables.

The events in a resource-driven simulation simply increment or decrement job counts and numbers of available resources. Very large and highly-congested queueing networks (with any number of jobs of any number of types at any countable discrete stages of processing) can be modeled this way with a relatively small, finite set of integers. Simulations with only a few simple events can model a wide variety of systems with different queue disciplines and job priorities, re-entrant flow, alternative or multiple resource requirements, batch processing, random job loss due to balking or reneging, and resource failures and repair. Of course, it is always possible to create and maintain records for some individual jobs for data collection or during critical stages of processing without resorting to tracking all of the jobs.

In addition to their simplicity, resource-driven simulations have a number of advantages over conventional job-driven process flow models that describe the paths of individual jobs. Resource-driven simulations of highly congested systems have been created that execute orders of magnitude faster than corresponding job-driven process flow simulations. Furthermore, the memory and speed advantages of a resource-driven simulation do not significantly degrade as the system becomes more and more congested; a situation where job-driven simulations often grind to a virtual standstill.

Another way of looking at these two classes of models is to think of job-driven models as “push” systems that seize resources whenever they are available and view resource-driven models as “pull” systems where resources seize waiting jobs when they become idle. The problem of simulation logic deadlock (unrelated to deadlock in a physical system) in large job-driven simulations, is virtually eliminated in resource-driven models.

4 ANALYTICAL MODELS

For some resource-driven queueing simulations, we can develop an analytical model directly from the event graph. Networks of such queues can also be modeled.

4.1 Analytical Model for a G/G/1 Queue Trajectory

We will start by specifying the trajectory (say, queue sizes and in special cases, job delay times) of N jobs processed in a G/G/1 queue as the solution to a linear program. The event graph for a G/G/1 queue is Figure 2 with a batch size of B=1 and the number of idle resources, S, initially equal to 1. Possible applications are discussed near the end of the paper. For the time being, we will simply view solving the linear program as an alternative to “running” a G/G/1 simulation.

For this system, a linear program that specifies the dynamic system trajectory is almost obvious. The non-negative decision variables in our linear program will be the event times. Here

A_i = the time of the ith Arrival event,
S_i = the time of the ith Start event, and
F_i = the time of the ith Finish event.

The objective will be to execute each event as soon as possible subject to the constraints imposed by the event graph,

$$\text{Min } Z = \Sigma(A_i+S_i+F_I).$$

Other objective functions will work for this model. In fact, we do not need to include the job arrival times in our objective function since they are not scheduled by conditional edges. Also, if we knew that the N jobs occurred in the same busy period, then the simple objective of minimizing the length of the busy period, F_N, might be used. Note that we do not specify that the i^{th} job to arrive is also the i^{th} job to start service - the queue discipline is, so far, implicit.

The edges of the event graph provide constraints on when events can be executed. For a given input process of arrival times and service times ($t_A(i)$, $t_S(i)$: i = 1,2,…n). Simple linear constraints are enforced by each edge in the event graph (G/G/1 $\in L$)

$$\begin{array}{ll} A_{i+1} - A_i = t_A(i) & \text{(Arrival-Arrival edge)} \\ F_i - S_i = t_s(i) & \text{(Start-Finish edge)} \\ A_i \le S_i & \text{(Arrival-Start edge)} \\ F_i \le S_{i+1} & \text{(Finish-Start edge)} \end{array}$$
$$A_i,\ S_i,\ \text{and } F_i \ge 0$$

The first two constraints merely state that time is non-negative. The third constraint states that the server cannot start its i^{th} job until at least i jobs have arrived. The final constraint states that the single server cannot process more than one job at a time. (We have not imposed any condition that jobs must be processed in any particular order.)

The analytic solution to this model is the dynamic system trajectory. If for example the queue discipline is a

FIFO queue, The sequence of customer waiting times is, $W_i=S_i- A_I$; these are surplus variables in our third constraint. These waiting times can be computed either after the model is solved or by adding it as a constraint.

Preliminary experiments using CPLEX and MINOS to solve this linear program indicate that it can take more time to generate the input data than it does to solve the optimization program for the system trajectory. Furthermore, the linear program may "run" this model faster than conventional simulation methodology.

4.2 Parallel Resources and Batch Processing

We next will illustrate the approach with the simulation of multiple parallel resources and batch processing in Figure 2.

To test the truth of edge conditions, an event counting point process is used. The number of times that event E has occurred by time t is given by the right-continuous event counting function.

$$C_E(t) = \lim_{\varepsilon\to 0} \max\{i; E_i \le t+\varepsilon\}.$$

We will use the well-known relationship between an event and its counting point process,

$$E_i \le t \Leftrightarrow C_E(t) \ge i,$$

This relationship simply enforces the forward progression of time: if the ith occurrence of event E is at or before time t then it must have occurred at least i times by time t.

The number of jobs in line at time t, Q(t), is equal to the number of Arrival events (incrementing Q by 1) that have occurred minus B times the number of Start events (each decrementing Q by B) that have occurred, or,

$$Q(t) = C_A(t) - B * C_S(t).$$

At any time in the simulation, Q(t) must be greater than zero. Constraints on Start event times need to be considered since this is the only event scheduled by conditional edges in this event graph. Consider the instant, S_i, at which the ith batch service starts, so that $C_S(S_i) = i$.

Then,

$$\begin{aligned} &0 \le Q(S_i) = C_A(S_i) - B * C_S(S_i)\\ &\Rightarrow C_A(S_i) \ge B * C_S(S_i)\\ &\Rightarrow C_A(S_i) \ge B*i\\ &\Rightarrow A_{B*i} \le S_i \end{aligned}$$

This constraint, $A_{B*i} \le S_{I,}$ simply says that, since jobs are processed B at a time, then B times as many jobs must have arrived as have started service.

Consider again the instant, S_i, when the ith service starts. For service to start, there must be at least one idle resource. The number of idle resources at any time is equal to the initial number of idle resources, R, less the number of Start events (decrementing resources) plus the number of Finish events (incrementing resources). Therefore, at the time of the ith Start event, S_i, there must be a non-negative number of idle resources,

$$\begin{aligned} &R + C_F(S_i) - C_S(S_i) \ge 0\\ &\quad\Rightarrow C_F(S_i) \ge C_S(S_i) - R\\ &\quad\Rightarrow C_F(S_i) \ge i - R\\ &\quad\Rightarrow F_{i-R} \le S_i \end{aligned}$$

This constraint, $F_{i-R} \le S_i$ simply says that, since there are only R resources, the number of Start events cannot exceed the number of Finish events by more than R. This constraint is enforced in the simulation by the edge from the Finish event to the Start event in Figure 2.

In general, the number of occurrences of events that decrement the availability of a limited resource can never exceed the number of events that increment that resource by more than the number of such resources. Sets of events in the resource-driven simulation that relate to limited resources will have such constraints even if they do not share an edge in the event graph.

To summarize: The event graph in Figure 2 translates into the following linear program

(Events occur as soon as feasible)

$$\text{Min: } \Sigma\,(A_i + S_i + F_i)$$

The two unconditional timed edges provide the constraints

$$A_{i+1} = A_i + t_A(i) \text{ and}$$
$$F_k = S_j + t_S(j) \text{ (plus assignment constraints)}$$

The assignment constraints referred to here are to insure that only one service time is used for each job.

The two conditional zero-delay edges provide the constraints

$$S_i \ge A_{Bi} \text{ and } S_i \ge F_{i-R}.$$

The subscripts on the last constraints reflect the bounds on number of resources, R, in our system and the Batch size. The queue length process, $Q(t) = C_A(t) - BC_S(t)$ can easily be computed from the trajectory regardless of the queue discipline.

5 POTENTIAL APPLICATIONS

A few applications of this methodology come to mind.

5.1 Formulating Scheduling Problems

The linear program for the multiple resource simulation becomes more interesting when the order in which J jobs are to be processed is to be determined. Translating the event graph into what now becomes a mixed integer scheduling program (IP) gives a formulation of the classical parallel server non-preemptive job scheduling problem which is in the class of NP hard problems. Nevertheless, solving this IP for small numbers of jobs gives us insights into possible efficient heuristics. To add generality, the order in which jobs arrive can also become a policy decision (say, to derive an order- release rule).

For the parallel resource event graph , if the objective function is changed to Min(F_N) added to binary decision variables for assigning the i^{th} job processing time to the j^{th} Start event then we have a formulation of the parallel resource scheduling problem. The solutions to several small random instances of the above scheduling problem consistently identified the Longest Processing Time Rule, known to be efficient for this problem (Pinedo 1995).

5.2 Sensitivity Analysis

The average waiting time for a busy period of N customers in our FIFO G/G/1 queue is given by $W=\Sigma(S_i-A_i)/N$. Viewing W as a surplus variable, we might be able to smooth its shadow cost and use this along with the shadow costs for the timed edge constraints to estimate the sensitivity of W to changes in service time or arrival time parameters. Analytical values for the derivatives of event delay times to their parameters might be used in a chain rule gradient estimator as done in Infinite Perturbation Analysis.

However, It might be more straightforward and practical to do the following:

1. Run the simulation in the usual manner;
2. Use the event times from the simulated sample path as an initial solution for the optimization program; and
3. Compute the sensitivity of the system's performance to finite differences in a parameter starting from this solution using the well-developed methods of mathematical programming.

6 CONCLUSIONS

In 1999 W. H. Fleming (Fleming 1988) stated in that "…there are no models of discrete event systems that are mathematically as concise or computationally as feasible as are differential equations for continuous variable dynamic systems." It is hoped that this paper moves event graphs a step closer to putting discrete event dynamic system modeling on the same footing as that of continuous systems as first conjectured by Yucesan (1989).

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. Discussions with Ilan Adler, Paul Hyden, and Philip Kaminsky were also helpful.

REFERENCES

Fleming, W. H., *Report on the panel of future directions in control theory mathematical perspective*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1988.

Pinedo, M., *Scheduling theory, algorithms, and systems*, Prentice Hall, New Jersey, 1995

Ross, S. M. *Introduction to probability models*, Academic Press, 1993.

Schruben, D and L. Schruben, *Graphical simulation modeling using SIGMA,* Custom Simulations, 2000.

Yucesan, E. Simulation graphs for design and analysis of discrete event simulation models, PhD thesis, School of Operations Research and Industrial Engineering, Cornell University, 1989.

AUTHOR BIOGRAPHY

LEE SCHRUBEN is a Professor in the Department of Industrial Engineering and Operations Research at the University of California, Berkeley. His research is on discrete event simulation modeling and analysis methodologies in a variety of applications including manufacturing, agriculture, health care, banking, and broadcasting. He holds a BS degree from Cornell, a MS degree from the University of North Carolina, and a Ph.D. from Yale University. Prior to joining the Berkeley faculty in 1999, Professor Schruben was in the School of Operations Research and Industrial Engineering at Cornell University where he was the Andrew S. Schultz Jr. Professor of Industrial Engineering. He has been the Distinguished Visiting Professor at SEMATCH and the Principal Investigator for Cornell's Center for Semiconductor Manufacturing Research. His current e-mail address is `<schruben@ieor.berkeley.edu>`.