Proceedings of the 1988 Winter Simulation Conference
M. Abrams, P. Haigh, and J. Comfort (eds.)

Simulation graphs

Lee W. Schruben
School of O.R.I.E.
Cornell University
Ithaca, N.Y. 14853

Enver Yucesan
School of O.R.I.E.
Cornell University
Ithaca, N.Y. 14853

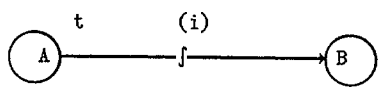
ABSTRACT

The Simulation Graphs presented in this paper are a mathematical formalization and extension of Event Graphs introduced by Schruben (1983) as a graphical representation of the event-scheduling approach in discrete-event simulations. Any simulation, indeed any computer program, can be modeled using a Simulation Graph. Finally, we present a graph theoretic approach to define equivalence of simulations.

1. INTRODUCTION AND BACKGROUND

The Simulation Graphs presented in this paper are a mathematical formalization of the Event Graphs introduced by Schruben (1983) as a graphical representation of the event-scheduling approach in discrete-event simulations.

Pictorially the vertices of an event graph represent state changes that are associated with the various events in the simulation. The edges of the graph represent the logical and temporal relationships between the vertices. For example, suppose the following edge is part of a simulation graph,



This edge is interpreted as follows: "whenever event A occurs, if condition (i) is true, then event B will be scheduled to occur t time units later." When t is equal to an infinitesimal increment, δt, the graph represents a differential equation simulator; when t can be a random variable, the graph represents a discrete-event simulator; when t has multiple spatial dimensions, the graph represents a finite-element simulator.

The elements of a simulation model are the state variables, events that change the values of state variables, and the relationships between the events. An event graph is a structure of the objects in a discrete-event system that facilitates the development of a correct simulation model.

Events are represented on the graph as vertices; each is associated with a set of changes to the state variables and has a partial ordering of execution priorities. In the computer, each vertex of the graph is a pointer to a string of expressions giving the state changes that result when the corresponding event occurs.

Relationships between events are represented in an event graph as directed edges between pairs of vertices. Each edge is a set of pointers to strings of logical and temporal expressions. Basically, the edges define under what conditions and after how much of a time delay an event will schedule another event to occur. Associated with each edge is a set of conditions that must be true in order for the edge origination event to schedule the termination event. Also associated with each edge is an expression or function that will give a delay time telling how long until the scheduled event will occur. There can be multiple edges between any pair of event vertices in the model; these edges can point in either direction. As a modeling convenience, there may also be cancelling edges in the graph.

1.1 The Definition of a Simulation Graph

A Simulation Graph is an ordered quadruple $G=(\mathcal{V}(G),\mathcal{E}_s(G),\mathcal{E}_c(G),\Psi_G)$, where $\mathcal{V}(G)$ is the vertex set of G, $\mathcal{E}_s(G)$ is the set of scheduling edges of G, $\mathcal{E}_c(G)$ is the set of cancelling edges of G and Ψ_G is the incidence function [Schruben and Yucesan, 1987]. We may then formally define a simulation model as the following structure of indexed sets:

$\mathcal{F} = \{f_i : \text{STATES} \rightarrow \text{OUTPUTS} \mid i \in \mathcal{V}(G)\}$, which is the set of state transitions associated with vertex (event) i;

$\mathcal{C} = \{c_{ij} : \text{STATES} \rightarrow \{0,1\} \mid (i,j) \in \mathcal{E}(G)\}$, which is the set of edge conditions; (also note that $\mathcal{E}(G) = \mathcal{E}_s(G) \cup \mathcal{E}_c(G)$)

$\mathcal{T} = \{t_{ij} \mid (i,j) \in \mathcal{E}(G)\}$, which is the set of edge delay times;

$\Gamma = \{\gamma_i \mid i \in \mathcal{V}(G)\}$, which is the set of event execution priorities; note that Γ is the set of nonnegative integers; we will also assume that smaller integers will correspond to higher priorities with 0 representing the highest execution priority;

S_i is the set of state variables possibly altered by event vertex i, $i \in \mathcal{V}(G)$;

E_i is the set of state variables involved in the conditions on the arcs emanating from vertex i, $i \in \mathcal{V}(G)$;

$\mathcal{L}$ is the list of scheduled events (events list);

τ is the global simulation clock.

The simulation also includes a scalar stochastic process of independent uniform random variables on (0,1).

2. SIMULATION GRAPHS AND TURING MACHINES

The Turing Machine is a simple mathematical model of a computer. Despite its simple structure, it captures all of the essential features of real computing machines. Hence, it is the accepted formalization of an effective procedure or an algorithm. This is partly a result of the Church's Thesis, which states: "Turing Machines are formal versions of algorithms and no computational procedure will be considered an algorithm if it cannot be rendered as a Turing Machine" [Lewis and Papadimitriou, 1981]. Note that this is not a theorem; it merely establishes the correspondence between an informal concept and a mathematical object. Nevertheless, the Turing Machine has the computing power of a digital computer and is equivalent to the general mathematical notions of computation. Thus, establishing the equivalence between Turing Machines and Simulation Graphs demonstrates the general computational power of the Simulation Graphs.

Following Hopcroft and Ullman (1979), we define a basic Turing Machine, TM, as consisting of a finite control, an input tape that is divided into cells and a tape head that scans one cell of the tape at a time. The tape has a leftmost cell, but is infinite to the right. Each cell of the tape may hold exactly one of a finite number of tape symbols.

In one move, depending on the symbol scanned by the tape head and the state of the finite control, TM changes state, prints a symbol on the scanned cell and moves its head left or right one cell.

Formally, TM is defined as:

$$M = (Q,\Sigma,T,\delta,q_0,B,F)$$

where

Q is the finite set of states,
T is the finite set of allowable tape symbols,
B is the symbol for blank, ($B \in T$),
Σ is the set of input symbols, ($\Sigma \subset T \setminus \{B\}$),
δ is the next move function, $\delta: Q \times T \rightarrow Q \times T \times \{L,R\}$
q_0 is the start state, ($q_0 \in Q$),
F is the set of final states, ($F \subset Q$)

There are many modifications of TM. Two-way infinite tape, multiple tracks, multiple tapes are among such modifications. These alterations are

mainly for convenience in particular applications and do not increase the computational power of TM. For equivalence theorems, see Hopcroft and Ullman (1979).

The equivalence between Turing Machines and Simulation Graphs is established in Schruben and Yucesan (1987). This result demonstrates that any simulation, indeed any computer program, can be modeled using a Simulation Graph.

3. EQUIVALENCE OF SIMULATION GRAPHS: A GRAPH THEORETIC APPROACH

3.1. Preliminary Definitions

In a Simulation Graph, an edge condition will be called simple if it consists of two arithmetic expressions connected by a relational operator. In other words, a simple edge condition is a relation. The arithmetic expressions may simply be a constant or a variable, whereas the relational operators are "less than," "less than or equal to," "greater than," "greater than or equal to," "equal to" and "not equal to." On the other hand, the edge condition will be called complex if it consists of two or more relations joined by Boolean operators AND or OR. For example, [QSIZE>0] is a simple condition, while [(QSIZE=0)AND(S=1)] is a complex edge condition.

Similarly, a vertex will be considered simple if there is at most one state variable change associated with it. In other words, vertex E is a simple event vertex if its execution alters the value of at most one state variable.

Otherwise, the vertex will be called a complex vertex. A vertex with no state variable changes will be referred to as the identity vertex.

Finally, a Simulation Graph G will be called an Elementary Simulation Graph and denoted G^E, if it contains only simple event vertices and the edge conditions are all simple.

Given a Simulation Graph G, one can always construct an associated Elementary Simulation Graph, G^E, by expansion. This is the process of replacing a single vertex with m state variable changes ($m \geq 1$) by m vertices in series, each with a single state variable change. It is also the process of replacing an edge with a complex condition by a series of identity vertices, each connected with simple edges.

A graph G can be thought of as a triple, $G = (\mathcal{V}(G), \mathcal{E}(G), \Psi_G)$, where $\mathcal{V}(G)$ is the vertex set of G, $\mathcal{E}(G)$ is the edge set of G, and Ψ_G is the incidence function [Bondy and Murty, 1976]. Since we are mainly dealing with directed graphs, $\Psi_G(e) = uv$ if e is an edge directed from vertex u to vertex v.

We will call graphs G and H identical and write $G = H$ if $\mathcal{V}(G) = \mathcal{V}(H)$, $\mathcal{E}(G) = \mathcal{E}(H)$ and $\Psi_G = \Psi_H$. Clearly, two identical graphs can be represented by the same diagram. However, it is still possible for graphs that are not identical to be represented by the same diagram. Such graphs are called isomorphic. More precisely, G and H are isomorphic, if there exist one-to-one and onto mappings (bijections)

$$\Theta : \mathcal{V}(G) \rightarrow \mathcal{V}(H)$$
$$\Phi : \mathcal{E}(G) \rightarrow \mathcal{E}(H)$$

such that $\Psi_G(e) = uv$ if and only if $\Psi_H(\Phi(e)) = \Theta(u)\Theta(v)$. The pair of mappings (Θ,Φ) is called an isomorphism [Bondy and Murty, 1976].

Note that isomorphic graphs form an equivalence class. Thus, we are not interested in the particular names of the vertices or edges, and distinguish between graphs only up to isomorphism.

For Simulation Graphs, we will adopt slight extensions of the above definitions. Recall that we have defined a Simulation Graph G by a quadruple $G = (\mathcal{V}(G), \mathcal{E}_s(G), \mathcal{E}_c(G), \Psi_G)$ [Schruben and Yucesan, 1987]. A simulation model is then

defined as a structure of indexed sets. Four of these sets are of particular importance to our construction here:

Γ - set of event execution priorities,

S_i - set of state variables possibly altered by event vertex i,

C - set of edge conditions,

$\mathcal{T}$ - set of edge delay times.

Also define, for a Simulation Graph, G,

$S(G) = \cup_{i \in V(G)} S_i$, set of state variables possibly altered by any event vertex.

Moreover, let $S'(G) \subseteq S(G)$.

Let $\Gamma(G)$, $S'(G)$, $C(G)$, $\mathcal{T}(G)$ and $\Gamma(H)$, $S'(H)$, $C(H)$, $\mathcal{T}(H)$ be the above defined sets associated with Simulation Graphs G and H, respectively.

Then, two Simulation Graphs G and H will be called <u>isomorphic</u> if there exist bijections:

$$\theta : V(G) \to V(H)$$
$$\nabla : \Gamma(G) \to \Gamma(H)$$
$$\Lambda : S'(G) \to S'(H)$$
$$\Omega : C(G) \to C(H)$$
$$\chi : \mathcal{T}(G) \to \mathcal{T}(H)$$
$$\Phi_s : \mathcal{E}_s(G) \to \mathcal{E}_s(H)$$
$$\Phi_c : \mathcal{E}_c(G) \to \mathcal{E}_c(H)$$

such that $\Psi_G(e) = uv$ if and only if $\Psi_H(\Phi_s(e)) = \theta(u)\theta(v)$ for all $e \in \mathcal{E}_s(G)$ and $\Psi_G(f)=xy$ if and only if $\Psi_H(\Phi_c(f))= \theta(x)\theta(y)$ for all $f \in \mathcal{E}_c(G)$. That is, the mappings $(\theta,\nabla,\Lambda,\Omega,\chi,\Phi_s,\Phi_c)$ form an isomorphism. Simply stated, names given to objects in a graph are <u>not</u> important since isomorphic graphs form an equivalence class.

Note that the isomorphism is defined over subsets of state variables whose values may be altered by the execution of any event. This provision allows for the modeler to focus on those state variables that are relevant in the scope of the simulation study.

3.2. Equivalent Simulation Graphs

<u>Definition :</u> Simulation Graphs G and H are <u>equivalent with respect to a subset of the state variables</u> if their Elementary Simulation Graphs G^E and H^E are isomorphic.

The above definition is important in at least two aspects: testability and utility. Testability refers to the ability to identify equivalent simulations possibly coded in different languages without actually running both models and comparing their outputs. Utility, on the other hand, refers to the ability to use the two models interchangeably once their equivalence is established.

The construction of Elementary Simulation Graphs as well as considerations for the validity of such procedures are presented in detail in Schruben and Yucesan (1988).

4. BIBLIOGRAPHY

[1] Bondy, J.A. and Murty, U.S.R. (1976) Graph Theory with Applications. North-Holland, New York.

[2] Hopcroft, J.E. and Ullman, J.D. (1979) Introduction to Automata Theory, Languages and Computation. Addison Wesley. Reading, MA.

[3] Lewis, H.R. and Papadimitriou, C.H. (1981) Elements of the Theory of Computation. Prentice Hall. Englewood Cliffs, N.J.

[4] Schruben, L.W. (1983) Simulation Modeling with Event Graphs. Communications of ACM. 26-11, 957-963

[5] Schruben, L.W. and Yucesan, E. (1987) On the Generality of Simulation Graphs. Technical Report No.773. SORIE, Cornell University, Ithaca, N.Y.

[6] Schruben, L.W. and Yucesan, E. (1988) Equivalence of Simulations: A Graph Theoretic Approach. Technical Report No.794. SORIE, Cornell University, Ithaca, N.Y.

Authors' Biographies

LEE W. SCHRUBEN is on the faculty of the School of Operations Research and Industrial Engineering at Cornell University. He received his undergraduate degree in engineering from Cornell University and a Masters degree from the University of North Carolina. His Ph.D. is from Yale University. Before going to graduate school, he was a manufacturing system engineer with the Emerson Electric Company in St.Louis, Mo. His research interests are in the statistical design and analysis of large scale simulation experiments. His consulting activities have been primarily focused in the area of manufacturing systems simulation. He is a member of ASA, ORSA, SCS and TIMS. He currently serves on several editorial boards for several journals.

Lee W. Schruben
School of O.R. & I.E.
Cornell University
Ithaca, N.Y. 14853
(607) 255-9139

ENVER YUCESAN is a Ph.D. student in the School of Operations Research and Industrial Engineering at Cornell University. He received his undergraduate degree in Industrial Engineering from Purdue University and a Masters degree in Operations Research from Cornell University. His research interests include simulation modeling, simulation output analysis and manufacturing systems.

Enver Yucesan
School of O.R. & I.E.
Cornell University
Ithaca, N.Y. 14853
(607) 255-9139