SCHOOL OF OPERATIONS RESEARCH
AND INDUSTRIAL ENGINEERING
CORNELL UNIVERSITY
ITHACA, NEW YORK 14853

TECHNICAL REPORT NO.842

April 1989

SIMULATION GRAPH DUALITY
A WORLD VIEW TRANSFORMATION FOR SIMPLE QUEUEING MODELS

by

Lee Schruben
and
Enver Yucesan

This work was sponsored in part by National Science Foundation Grant ECS 8810517.

ABSTRACT

Planar graphs play an important role in real world applications, partly due to the fact that some practical problems can be efficiently solved for planar graphs while they are intractable for general graphs. Simulation Graph Models of simple queueing systems are planar graphs. Their geometric duals can then be constructed. In the context of queueing models, this dualization process represents a transformation between the event scheduling and the activity scanning world views in discrete event simulations. The so-called primal-dual pair of models provides an alternative but equivalent representation of these stochastic systems.

Key words: Planarity, geometric dual, world views

1. INTRODUCTION

A graph is said to be *planar* if it can be drawn on the plane so that no two edges intersect except possibly at a vertex. Planar graphs arise quite naturally in real world problems. For instance, the design of integrated circuits requires knowing when a circuit may be embedded in a plane. Determining isomorphism of chemical structures is simplified if the structure is planar. Planar graphs play an important role in these applications since some practical problems can be efficiently solved for planar graphs while they are intractable for general graphs [Nishizeki and Chiba, 1987].

A property of planar graphs is that they have *geometric duals*. In this paper, we show how this property can be used together with Simulation Graphs to define a transformation between two world views in discrete event simulations. A *Simulation Graph* is a network used to construct and analyze discrete event simulation models [Schruben and Yucesan, 1987]. On this network, each vertex represents state changes associated with a system event, while each directed edge depicts the logical and temporal relationships between these events. *World views* refer to system structuring approaches commonly used in discrete event simulation modeling [Schruben, 1983].

The relevant graph theoretic concepts are presented in the Appendix. The remainder of the paper is organized as follows: Section 2 discusses the world views in discrete event simulations. Simulation Graphs are reviewed in Section 3. Geometric duality in planar graphs is covered in Section 4. Section 5 discusses this duality in the context

of Simulation Graphs and exhibits a world view transformation for simple queueing models. The implications of this result along with some concluding comments are included in Section 6.

2. WORLD VIEWS

In discrete event simulations, world views provide alternate approaches to organizing a specification of model behavior. These modeling perspectives are commonly called *event scheduling*, *activity scanning* and *process interaction*. Even though each of these world views is supported by one or more simulation programming languages, no generally accepted definitions exist for any of the approaches. This is partly due to the fact that this classification scheme is neither mutually exclusive nor collectively exhaustive. Nevertheless, "world views are important, because they 'drive' the development of methodologies for model formulation and significantly impact the form and content of the models themselves. World views provide a foundation, a platform, for the development of specification languages and other types of computer assistance and support of the formulation of simulation models" [Burns and Morgeson, 1988].

The definitions presented here are from [Overstreet, 1987]. He characterizes each world view as emphasizing a different type of "locality." The latter is defined as "that property when all relevant parts of a program are found in the same place."

Event scheduling emphasizes locality of *time*. Each event routine in a model specification describes a collection of actions which may normally all occur in one instant, that is, with no time advance.

Activity scanning emphasizes locality of *state*. Each activity routine in a model specification describes a collection of actions which will occur once the model achieves a specified state.

Process interaction emphasizes locality of *object*. Each process routine in a model specification describes all actions taken by one model object or, more precisely, one class of model objects.

We next discuss the event scheduling and activity scanning world views in more detail. Our exposition is based on [Balci, 1988].

2.1 Event Scheduling Approach

In this world view, an event is the major focus of modeling a system. Within this approach, the objects in a system are identified first and described through state variables. Next, what causes changes in the values of state variables are defined as an event. Each event is usually implemented as a separate subroutine or procedure in the computer program.

The execution of such a model is carried out using an *events list*, a list of events scheduled to occur in the simulated future, and a global simulation clock. Each entity of the events list is called an *event record*, which contains information about the event type, event time and other attributes associated with the particular event.

The execution starts with initialization, where initial values of state variables are established and appropriate event are scheduled. The process continues by advancing the simulation clock to the time of the next (most imminent) event and the execution of the associated event routine. The latter may alter the values of state variables and may schedule further events by inserting appropriate event notices into the events list. The logic of the event scheduling approach is summarized in Figure 1.

FIGURE 1. EVENT-SCHEDULING WORLD VIEW

2.2 Activity Scanning Approach

This approach is attractive for models where the number of possible events is small, but the conditions under which the events occur are very complex. From this perspective, the approach is similar to the rule-based programming used in Artificial Intelligence, in which rules are specified for controlling the performance of predetermined sets of operations [Balci, 1988].

Within this world view, the modeler describes an activity in two parts: (i) a condition which must be satisfied for the activity to take place, (ii) the operations of the activity performed upon the satisfaction of the activity's condition.

The logic of execution is as follows: the initializations include the assignment of initial values to state variables. The simulation clock is updated in fixed increments. That is, in Phase 1, time is advanced from t to $(t+\delta t)$. Phase 2 is then conducted with a clock time of $(t+\delta t)$. At that point, the conditions of activities are tested in the order of activity priorities. If an activity's condition is satisfied, the associated operations are performed. A single scan may not be sufficient since some actions performed may cause the satisfaction of previously unsatisfied conditions. This requires the restart of Phase 2. All the conditions must be repeatedly tested until no condition is satisfied at the current clock time. The activity scanning approach is summarized in Figure 2.

FIGURE 2. ACTIVITY SCANNING WORLD VIEW

3. SIMULATION GRAPHS

The elements of a discrete event simulation model are state variables that describe the objects in a system, events that change the values of state variables, and the relationships between events. A Simulation Graph is a structure of the objects in a discrete event system that facilitates the development of correct simulation models.

Events are represented on the graph as vertices. Each vertex is associated with a set of changes to state variables.

Relationships between events, on the other hand, are represented as directed edges between pairs of vertices. Each edge depicts under what conditions and after how much of a time delay an event will schedule or cancel another event. More specifically,

indicates that "t time units after the occurrence of event A, event B will be scheduled to occur with parameter string k=j, provided that condition (i) holds at the time event A occurs." The parameter string carries information pertaining to a particular event instance as well as the execution order priority of that event to break any possible time ties. These strings can be passed in a model, through vertex and edge attributes.

More formally, a Simulation Graph is defined as an ordered quadruple, $g = (V(g), E_s(g), E_c(g), \Psi_g)$, where $V(g)$ is the vertex set of g , $E_s(g)$ is the set of scheduling edges of g , $E_c(g)$ is the set of

cancelling edges of g , and Ψ_g is the incidence function. We then define the objects in a simulation model as the following ordered sets:

$\mathcal{F} = \{ f_v : v \in V(g) \}$, set of state transitions,

$\mathcal{C} = \{ C_e : e \in \mathcal{E}_s(g) \cup \mathcal{E}_c(g) \}$, set of edge conditions,

$\mathcal{T} = \{ t_e : e \in \mathcal{E}_s(g) \}$, set of edge delay times,

$\Gamma = \{ \gamma_e : e \in \mathcal{E}_s(g) \cup \mathcal{E}_c(g) \}$, set of event execution order

priorities.

A *Simulation Graph Model* is then defined as:

$$\mathcal{S} = (\mathcal{F}, \mathcal{C}, \mathcal{T}, \Gamma, g).$$

The key idea here is that a Simulation Graph specifies the relationships between the elements of the sets of objects in a simulation model.

We conclude this section with a simple example to illustrate the concepts discussed above. For a complete treatment of Simulation Graphs, the reader is referred to [Schruben and Yucesan, 1987].

Example: Single Server Queueing System

We will develop a Simulation Graph Model of a single server queueing system. Suppose that customers arrive into the system every t_A time units and it takes the server t_S time units to attend to each customer. The state variables used in this model are:

Q , the number of customers waiting for service, and

S , the status of the server (0 = busy, 1 = idle).

The edge conditions for the model are:

(i) (The server is idle) $S = 1$,

(ii) (Customers waiting to be served) $Q > 0$.

The event descriptions are presented in Table 1.

TABLE 1

<u>Event Type</u>	<u>Event Description</u>	<u>State Changes</u>
ARV	Customer arrival	$Q = Q + 1$
BGN	Beginning of service	$S = 0$ $Q = Q - 1$
END	End of service	$S = 1$

The associated Simulation Graph is presented in Figure 3.

FIGURE 3. SINGLE-SERVER QUEUEING MODEL

4. GEOMETRIC DUALITY

Recall that a graph is said to be planar if it can be drawn on the plane so that its edges intersect only at their ends. A planar graph embedded in the plane is referred to as the plane graph. The "regions" defined by a plane graph G are called the faces of G , the unbounded region being the exterior face. Given a plane graph G , the geometric dual G^* is constructed as follows:

Step 1: Place a vertex in each face of G , including the exterior face.

Step 2: If two faces of G have an edge e in their common boundary, join the vertices of the corresponding faces by an edge e^* crossing only e .

The result may be a plane graph with loops or multiple edges. The graph G^D for which G^* is a plane graph is said to be the dual of G . The plane graph G is not unique; hence, its dual is not unique either.

Figure 4 illustrates the dualization procedure.

When G is connected, the dual of the dual of G is isomorphic to G , that is $G^{**} \cong G$. The following lemma from [Nishizeki and Chiba, 1987] summarizes the relationship between G and G^* .

Lemma: Let G be a connected plane graph with n vertices, m edges and f faces; let G^* be the geometric dual with n^* vertices, m^* edges and f^* faces. Then,

$$n^* = f,$$

$$m^* = m,$$

$$f^* = n.$$

FIGURE 4. CONSTRUCTION OF G^* FROM G

5 DUALITY IN SIMULATION GRAPHS

5.1 Planarity

Before exploring the concept of duality within the context of Simulation Graphs, we will establish that these graphs are planar for models of simple queueing systems (e.g., G/G/s).

Theorem: Simulation Graphs for models of simple queueing systems are planar.

To prove this theorem, we need the following result from [Bondy and Murty, 1976]: *A graph is planar if and only if each of its blocks is planar.* The proof is presented next.

Proof: The following is a Simulation Graph for a generic queueing system such as a G/G/s model:

This generic Simulation Graph can represent many simple queueing systems. For example, different customer classes can be handled through the use of event attributes [Schruben and Yucesan, 1987]. That is, the arrival of a customer of type j can be represented by the event vertex $A(j)$. The reader can easily verify that balking and blocking can also be incorporated without losing planarity. The blocks of this graph

(ignoring the direction and the conditions on the edges) are given below:

Since every graph is a union of its blocks and since all of the blocks, B₁, B₂ and B₃, are planar, it then follows from the above result that Simulation Graphs for simple queueing models are planar.

Furthermore, it is possible to determine whether a given Simulation Graph is planar. There are a number of algorithms for establishing the planarity of graphs. For instance, one such algorithm is presented in [Hopcroft and Tarjan, 1974].

From this point on, we will only be considering Simulation Graph Models of simple queueing systems.

5.2 Simulation Graph Duality

Since Simulation Graphs are planar, their dual can be constructed as described in Section 4. Note that the dual of a Simulation Graph is also a Simulation Graph. We will refer to $\mathcal{G}$ as the *primal Simulation Graph* or, simply, the *primal*; and $\mathcal{G}^D$ will be referred to as the *dual Simulation Graph* or, simply, the *dual*.

In this section, we will establish the fact that the primal Simulation Graph represents the event scheduling world view whereas the dual Simulation Graph represents the activity scanning world view. Hence, the process of constructing g^D from g really represents a transformation between these world views.

In the dualization procedure, we will assume that the Simulation Graph does not contain any loops; that is, there are no edges that originate and terminate at the same event vertex. This is not a restrictive assumption since any loop of the form:

can be replaced by a directed cycle of the form:

In this construct, the dummy event A_1 is always scheduled with the highest execution priority ($\gamma=0$) and no time delay. Figure 5 shows the single server queueing model under this convention.

It is also assumed that the Simulation Graph does not have any cancelling edges, since it has been shown that these edges are merely a modeling convenience [Schruben and Yucesan, 1987]. The dualization procedure is discussed next.

FIGURE 5. SINGLE-SERVER QUEUE REDRAWN

Step 0: Replace all the loops, if any, in the Simulation Graph with a directed cycle as described above.

Step 1: Ignore the directions on the edges and construct the geometric dual as before. The vertex in g^D , corresponding to the exterior face of g , is a special vertex. Label it "ESF" for "Event Scheduling Function."

Step 2: Assign Edge Directions: There must be one edge directed from the ESF vertex to every other vertex in g^D . All of the remaining edges should point in the opposite direction; that is, from a given vertex to ESF. Also note that the ESF vertex will always have a self scheduling edge.

Step 3: Edge Delay Times: All edges directed from ESF to other vertices have zero delay times. The delay times in g^D will be assigned to those edges incident into the ESF vertex. The delay time will be equal to the sum of the delay times on the edges comprising the boundary of the corresponding face in g .

Step 4: Edge Conditions: In a Simulation Graph Model, there are two basic types of events. Events that are scheduled to occur unconditionally at specific instants in simulated time will be called *time-dependent* events or *t-events*. There are also those events which are scheduled only if the associated conditions are satisfied. These will be referred to as *conditional* events or *c-events*.

In g^D , a new state variable is defined to condition the actions associated with a t-event in g . The new state variable will basically denote the execution time of the associated activity. (An example will soon follow.)

No new state variables are defined in g^D for activities corresponding to c-events in g . A condition, however, is assigned to the edge that is directed from ESF to the particular vertex. This condition is a complex one constructed by combining the edge conditions on the boundary of the corresponding face in g through the Boolean operator AND.

Step 5: State Changes: The state changes associated with c-events in g directly carry over to the corresponding activities in g^D . For the activities in g^D corresponding to t-events of g , there is one additional state variable change: the update of the activity's execution time.

Step 6: Attribute Passing: The values of state variables are altered upon the execution of an activity. The new values for the state variables should be transmitted back to the ESF vertex using the attribute lists of the edges directed into ESF.

Step 7: Priorities: Priorities must be assigned by the modeler for activity execution (that is, condition checking).

Step 8: Subdivision of ESF: The ESF vertex is subdivided into two vertices connected by a directed edge of zero time delay. The new vertices are called "Phase 1" and "Phase 2" respectively. The ESF vertex is subdivided in this manner to conform to the activity scanning logic. More specifically, the ESF vertex

is replaced by:

Here, Δt represents the fixed time increment that updates the simulation clock. Both of these vertices are always scheduled with the highest execution priority in g^D .

The dualization procedure is illustrated on the single server queueing model.

Example: (Revisited):

Figure 6 depicts the Simulation Graph for the single server queueing model along with its geometric dual. In the original (primal) model, since the arrival event (ARV) is a t -event, a new state variable, ARVT, is defined to denote the execution time of the arrival activity in g^D . The edge conditions used in the dual are as follows:

- (i) (Time for an arrival) $ARVT = \tau$ (τ denotes the simulation clock),
- (ii) (Customers are waiting AND the server is idle) $(Q > 0) \ \& \ (S = 1)$.

The associated activities are described in Table 2.

TABLE 2

<u>Activity Type</u>	<u>Activity Description</u>	<u>State Changes</u>
PH.1	Phase 1: Time Scan	
PH.2	Phase 2: Condition Scan	Attribute List: $Q, S, ARVT$
ARV	Arrival Process	$Q = Q + 1$ $ARVT = \tau + t_A$
SVC	Service Process	$S = 0$

FIGURE 6. q AND q^D

$$Q = Q - 1$$

9 is presented in Figure 7. Note that the execution of the dual model directly follows the activity scanning logic depicted in Figure 2.

FIGURE 7. THE DUAL MODEL

6. CONCLUDING REMARKS

Within the context of simple queueing models, the dualization of Simulation Graphs represents a transformation between the event scheduling and the activity scanning world views. Hence, the primal-dual pair provides the modeler with alternative but equivalent representations of queueing systems.

It is conjectured that this primal-dual relationship is valid not only for simple queueing systems, but for any discrete event dynamical systems in general.

As Hooper (1986) notes: "for more than a quarter of a century, simulationists in the U.S. and in Britain have been on 'divergent paths' as to world views (i.e., strategies). It does not appear that we are likely to alter these courses in the foreseeable future. However, we can certainly benefit from a good understanding of the strategies that are in use." It is hoped that the construction presented in this paper provide a useful first step in bridging these paths.

REFERENCES

- [1] Balci, O. (1988) *The Implementation of Four Conceptual Frameworks For Simulation Modeling in High-Level Languages* Proceedings of the 1988 Winter Simulation Conference (Abrams, Haigh and Comfort, eds.) San Diego, CA.
- [2] Bondy, J.A. and Murty, U.S.R. (1976) *Graph Theory with Applications* North-Holland. New York.
- [3] Bratley, P. Fox, B. and Schrage, L. (1987) *A Guide to Simulation* (2nd Edition) Springer-Verlag. New York.
- [4] Burns, J.R. and Morgeson, J.D. (1988) *Object-Oriented World View for Intelligent, Discrete, Next Event Simulation Management* Science Vol. 34.12. pp. 1425-1440
- [5] Hooper, J.W. (1986) *Activity Scanning and the Three-Phase Approach Simulation* Vol.47.5
- [6] Hopcroft, J. and Tarjan, R. (1974) *Efficient Planarity Testing* Journal of the ACM Vol. 21.4 pp. 549-568
- [7] Lawler, E. (1976) *Combinatorial Optimization: Networks and Matroids* Holt, Rinehart and Winston. New York.
- [8] Nishizeki, T. and Chiba, N. (1987) *Planar Graphs: Theory and Algorithms* North-Holland Mathematics Studies #140
- [9] Overstreet, C.M. (1987) *Using Graphs to Translate Between World Views* Proceedings of the 1987 Winter Simulation Conference (Thesen, Grant and Kelton, eds.) Atlanta, GA.
- [10] Schruben, L.W. and Yucesan, E. *On the Generality of Simulation Graphs* Technical Report #773, School of OR&IE, Cornell University. Ithaca, New York.

[11] Suri, R. (1984) *A General Theory for Infinitesimal
Perturbation Analysis of Discrete Event Simulations* IEEE Trans.
Autom. Control

APPENDIX: RELEVANT CONCEPTS FROM GRAPH THEORY

Graph theory terminology is far from being uniform. Thus, the main purpose of this Appendix is to provide a comprehensive review of some relevant graph theoretic concepts. Our exposition is based on [Bondy and Murty, 1976], [Lawler, 1976] and [Nishizeki and Chiba, 1987].

A graph G is an ordered triple $(V(G), E(G), \Psi_G)$, where $V(G)$ is the set of vertices, $E(G)$ is the set of edges and Ψ_G is the incidence function, that associates with each edge a pair of not necessarily distinct vertices of G . If the pair of vertices is ordered, then the graph is said to be directed. If $\Psi_G(e) = (u, v)$, then the vertices u and v are said to be adjacent. We will let n denote the number of vertices of G (cardinality of $V(G)$), and m represent the number of edges of G (cardinality of $E(G)$). A graph H is said to be a subgraph of G , if $V(H) \subseteq V(G)$, $E(H) \subseteq E(G)$ and Ψ_H is the restriction of Ψ_G to $E(H)$.

A v_0-v_1 walk, W , is an alternating sequence of vertices and edges of G , $v_0 e_1 v_1 \dots e_{l-1} v_l$, beginning and ending with a vertex, in which e_k is incident only on v_{k-1} and v_k ($1 \leq k \leq l$). If the vertices are distinct, then W is called a path. W is closed if $v_0 = v_l$. A closed path containing at least one edge is a cycle. Two distinct vertices u and v are said to be connected if there is a path between u and v . A graph G is connected if all pairs of vertices are connected. A component of a graph is a maximal connected subgraph. A cut of a graph G is a set of edges of G whose removal increases the number of components. A cut set is a minimal cut; that is, it is the smallest possible set of edges with the given property. Similarly, a cut vertex is a vertex whose deletion increases the number of components. A

connected graph with no cut vertices is called a *block*. A *block of a graph* is a subgraph that is a maximal block; that is, it is the largest possible subgraph with the given property. Also note that every graph is the union of its blocks.

A graph in which every pair of distinct vertices are adjacent is called a *complete graph*. The complete graph on n vertices is denoted by K_n . A *bipartite graph* is one whose vertex set can be partitioned into two subsets X and Y so that each edge has one end in X and the other in Y . A complete bipartite graph, denoted by $K_{p,q}$, is a simple bipartite graph in which each vertex of X is adjacent to each vertex of Y .

A graph G' is said to be a *subdivision* of graph G , if G' is obtained from G by subdividing some of the edges, that is, by replacing the edges with paths having at most their endvertices in common.

A *planar graph* is a particular graph that can be drawn on the plane so that no two edges intersect geometrically except at a vertex at which they are both incident. A graph is planar if it has an embedding in the plane. The planar embedding of a planar graph is called a *plane graph*. The plane graph of G need not be unique.

A plane graph partitions the rest of the plane into a number of connected regions. The closures of these regions are called the *faces* of the graph. Each plane graph has exactly one unbounded face, called the *exterior face*. We will let f denote the number of faces of a plane graph. Next, we state some of the more significant results without proof:

1. If G is a connected plane graph, $n-m+f = 2$. This is known as Euler's formula.

2. If G is a simple planar graph with $n \geq 3$ vertices and m edges, then $m \leq 3n - 6$.

3. K_5 and $K_{3,3}$ are nonplanar.

4. If G is nonplanar, then every subdivision of G is nonplanar.

5. If G is planar, then every subgraph of G is planar.

The earliest characterization of planar graphs was given by Kuratowski. His theorem states that a graph is planar if and only if it contains no subdivision of K_5 or $K_{3,3}$. Although elegant, Kuratowski's theorem is not useful as a practical test of planarity. There are, however, numerous published planarity algorithms. For one such algorithm and a brief discussion of other algorithms, the reader is referred to [Hopcroft and Tarjan, 1974].