Proceedings of the 2007 Winter Simulation Conference
S. G. Henderson, B. Biller, M.-H. Hsieh, J. Shortle, J. D. Tew, and R. R. Barton, eds.

ALTERNATIVE THREAD SCORING METHODS IN QUALITATIVE EVENT GRAPHS

Ricki G. Ingalls

School of Industrial Engineering and Management
322 Engineering North, Oklahoma State University
Stillwater, OK 74078, U.S.A.

Douglas J. Morrice

Department of Information, Risk, and Operations Mgmt.
The University of Texas at Austin
Austin, TX 78712, U.S.A.

ABSTRACT

Event Graphs (EGs) and Simulation Graph Models provide a powerful and general modeling framework for discrete event simulation. Qualitative Event Graphs (QEGs) extend the EG framework to a qualitative approach to discrete-event simulation. In QEG, the uncertainty in event execution times is represented by a closed interval in the set of real numbers. When two or more event execution intervals overlap, multiple event execution sequences or threads result. This leads to simulation output in the form of multiple threads. In general, the number of threads can explode exponentially making output difficult to analyze. In this paper, we introduce three scoring methods to rank the threads on the relative likelihood of their event execution sequences. We discuss the assumptions of these methods along with their advantages and disadvantages. Depending on the needs of the user, scoring and ranking could help eliminate the need to execute some threads and cut the execution time of the simulation.

1 INTRODUCTION

Computer simulation is a flexible modeling technique used to solve problems in business, engineering, the physical sciences, and the social sciences. A computer simulation model is a computer program designed to characterize the behavior of an actual system. A number of different approaches to computer simulation modeling exist. In the physical sciences and engineering, a system is often modeled and simulated by a set of differential or difference equations. This approach provides very precise information about the behavior of the system as it evolves through time. It is often referred to as continuous time simulation. The continuous time paradigm has been abstracted or modified in a number of different ways to produce other approaches to simulation modeling.

One popular abstraction used extensively in engineering and business applications is called discrete-event simulation. Discrete-event simulation provides an efficient approach to modeling systems in which the state of the system changes at discrete points in time called events. To model such systems, the clock of the simulation model is incremented asynchronously through time by proceeding from one event to another. The behavior of the system is not monitored continuously through time because it is assumed that nothing of interest happens between events.

Another abstraction of the continuous time approach is called qualitative simulation or qualitative physics. Qualitative simulation was originally developed in the physical sciences (Forbus 1988), but more recently has found application in economics and business (Hinkkanen et al. 1993). The qualitative approach is useful when the level of knowledge about the system being modeled is imprecise. In fact, qualitative simulation is designed to represent whatever level of knowledge is available. For example, variables describing the state of a system might be represented in a qualitative simulation model as simply increasing, decreasing or constant with respect to time if no other information is available. Inferences derived from the results of a qualitative simulation model, although less precise, are often considered more general and robust since these inferences do not rely on precise and perhaps faulty assumptions.

Discrete-event simulation suffers from many of the same problems that motivated the pioneers of qualitative simulation. The amount of detail needed in a simulation often is overwhelming. Statistical distributions used in the simulations often are based on incomplete information or the intuition of the model builder. Because of the very nature of discrete-event simulation, appropriate levels of abstraction often are difficult to determine and justify. These problems make discrete-event simulation models difficult to build and to verify. The simulation expert can usually discredit a simulation by criticizing the input distributions, the appropriate level of detail, etc. Decision makers often consider simulation impractical because of lengthy analysis times and the difficulty of evaluating alternatives.

In Ingalls et al. (2000), QEGs were proposed and developed for qualitative discrete-event simulation. In this implementation, closed intervals in $\Re$ were used to represent event execution time uncertainty. The EG modeling framework was used because they provide a simple yet general representation of a discrete-event simulation (Schruben 1983, Som and Sargent 1989, Yücesan 1989). One problem for the QEG method is that the number of threads or event execution sequences can explode exponentially. This thread explosion makes it difficult to get meaningful information from the output. In this paper, we propose three scoring methods to rank the threads based on the relative likelihood of their event execution sequences. We discuss the assumptions of these methods along with their advantages and disadvantages. Such scoring methods are intended to provide the user with some means of focusing on the more likely scenarios and ignoring or perhaps even eliminating less likely scenarios saving on simulation execution time.

The remainder of the paper is organized in the following manner. Section 2 provides a brief description of the EG implementation. Section 3 describes the basic problem of multiple thread generation by a QEG. Additionally, we propose three scoring methods in this section. Section 4 illustrates the scoring methods on a PERT network example. Section 5 provides concluding remarks.

2 EVENT GRAPH IMPLEMENTATION

Ingalls et al. (2003) extended the Event Graph Implementation of Schruben, Yücesan, and others by eliminating the canceling edge and replacing it with the *edge execution condition.* This framework is the framework used in this paper. The basic construct of the event graph with the edge execution condition is shown in Figure 1. The nodes labeled A and B represent events and the edge specifies that there is a relationship between the two events. The construct is interpreted as follows: "if condition (i) is true at the instant event A occurs, then event B will be scheduled to occur *t* time units later. Event B will be executed *t* time units later with the state variables in array n set equal to the values in array k if condition (j) is true *t* time units later."

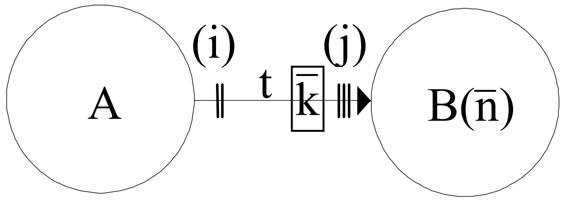

Figure 1: Event graph with canceling edges

3 CHARACTERIZING A THREAD

3.1 The Basic Problem

As an example of this problem, lets assume that we have a simulation that has only three events, A, B, and C. We are uncertain what order A, B, and C might occur because their execution times overlap. This set of events is called a *non-deterministically ordered set (NOS)*. Each event in the set is a *non-deterministically ordered event (NOE)* because the order of the set cannot be determined (Ingalls et al. 2000). Let us assume that the events overlap as seen in Figure 2, where event A can occur any time in the interval [2,5], event B can occur any time in the interval [1,6] and event C can occur any time in the interval [3,4]. Obviously, there are 6 sequences in which these 3 events can occur: ABC,

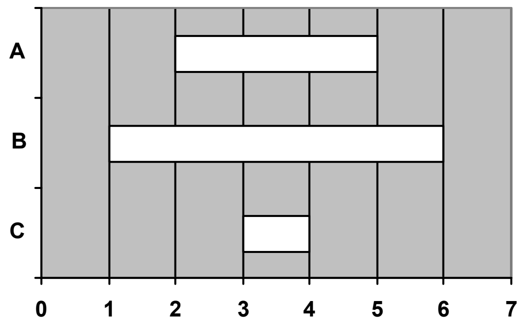

Figure 2: Example of a NOS

ACB, BAC, BCA, CAB, and CBA. The question that we are addressing in this paper is what sequence of events is more "likely". In Figure 3, we see A and B make up the intervals in the NOS. C is not in the NOS because it cannot possibly be the next event executed because A must occur before C. In qualitative simulation, there is a basic assumption that we do not know what type of distribution underlies the intervals. Although to use the term "likely" in

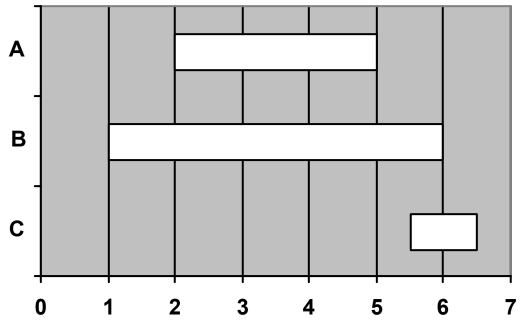

Figure 3: A and B make up the NOS

a qualitative simulation is not technically correct, in order to rank the threads, we must make minimal assumptions about the intervals in order to score them. One of our methods is more qualitative, in that we rank the intervals by the midpoint of the interval. Another uses the earliest midpoint as a basis and determines a relative score. The third method introduces a minimal statistical assumption, namely that the interval is uniformly distributed. With this assumption, we can calculate a score based on probabilities. In all three cases, the score is carried forward in the thread.

3.2 The Midpoint Ranking and Midpoint Multiple Methods

Let E_n, $n = 1, 2, \ldots, N$ ($N \geq 2$), denote events with respective execution intervals $[L_n, U_n]$ that overlap. Let M_n represent the midpoint of the interval $[L_n, U_n]$ for $n = 1, 2, \ldots, N$. We rank the intervals based on their midpoints. Let $Rank(M_n)$ denote the rank of M_n. We use the following two approaches for assigning $Rank(M_n)$:

(i) A simple numerical ordering of $1, 2, \ldots, N$. We will call this approach the *Midpoint Ranking Method.*

(ii) A ranking defined by

$$Rank(M_n) = \frac{M_n - \min\limits_{1\leq n\leq N}(L_n)}{\min\limits_{1\leq n\leq N}(M_n) - \min\limits_{1\leq n\leq N}(L_n)}. \qquad (1)$$

We will call this approach the *Midpoint Multiple Method.* The first ranking approach is simple to compute requiring the computation of a midpoint for each interval, a sort, and then the assignment of a numerical ranking. Approach (ii) requires the additional computing of expression (1). However, in addition to the order of the ranking, this approach provides a measure of the relative magnitude of the midpoints and therefore more discrimination on which event is likely to execute first.

The ranks are used to score each thread in the simulation. When N overlapping event intervals occur, each event spawns a new thread in which the event executes first. To the thread where E_n executes first, assign $Rank(M_n)$. Let S represent the NOS of N events with overlapping intervals, i.e., $S = \{E_1, E_2, \ldots, E_N\}$. Additionally, let A_e (B_e) be the set of events scheduled by the execution of E_n whose intervals overlap with the remaining N-1 events. The set A_e (B_e) may be empty if no such events are scheduled. The variable T is the current score on the thread. The algorithm for calculating all rankings on threads spawned by N (≥ 2) overlapping events can be stated recursively:

```
Procedure ComputeRank(N, S, T)
    For n = 1, 2, ..., N
        Compute $M_n$
        Calculate $Rank(M_n)$
        Calculate $A_e$ and $B_e$
        If ($N - 1 + |A_e| - |B_e| \geq 2$) then
            $S = S + A_e - E_n - B_e$
            $T' = T + Rank(M_n)$
            ComputeRank($N - 1 + |A_e| - |B_e|$, S, T')
        End If
    End For
End Procedure ComputeRank
```

At the end of the simulation, the score for each thread is computed by summing the ranks of all comparisons along the thread and dividing through by the total number of comparisons on the thread.

To illustrate the above algorithm, consider the example in Figure 2 with N = 3. To simplify the illustration, assume that A_e and B_e are empty sets at each stage. On the first pass through the algorithm, both ranking schemes would assign a rank of one to the threads spawned by each event because all three events have the same midpoint. If event A executes first, then the interval for B is reduced to [2,6]. Comparing this with C on the interval [3,4], the thread spawned by C gets assigned a rank of 1 by both approaches and B gets assigned a ranks of 2 and 1.33 by approaches (i) and (ii), respectively. Since N - 1 = 1, no further rankings are required along these threads. Thus, the sequence A before B before C gets assigned a score of 1.5 ((1+2)/2) and A before C before B gets assigned a score of 1 ((1+1)/2) by approach (i). Approach (ii) assigns the same sequences 1.165 ((1+1.33)/2) and 1 ((1+1)/2), respectively. Similarly, sequences B before A before C, B before C before A, C before A before B, and C before B before A are assigned scores 1(1), 1(1), 1(1), and 1.5 (1.25), respectively, by approach (i) (approach (ii)).

3.3 The Uniform Distribution Method

The idea behind this scoring methodology is that for every NOS, assign a score to an event which is the probability of that event being the next event to be executed. We assume, for the purpose of the scoring, that the event execution is uniformly distributed over the possible interval. This uniform distribution assumption is used because it requires the least amount of information to construct. Said another way, it is the probability measure that deviates the least from the qualitative simulation approach which assumes that the event could occur anywhere within the given interval.

The concept of the possible interval is important. In our example in Figure 2, let us assume that event A has been executed first. That leaves events B and C still to be executed and the current simulation time is [2,4]. For event C, it can still be executed in time interval [3,4]. However, event B can no longer be executed in interval [1,6]. The possible interval for B is now [2,6] since the clock must have already advanced to at least time 2 because A has already executed. For each of the methods presented in this paper, B and C would be compared using the intervals [2,6] and [3,4], respectively.

3.3.1 Thread Scoring Using the Uniform Distribution

For any *n* events in a NOS, each event has a possible execution time which is a qualitative interval. If we assume that the possible execution time is uniformly distributed with a minimum of a_n and a maximum of b_n, then the density function for possible execution time of the n^{th} event in the set is:

$$f_n(x) = \begin{cases} \dfrac{1}{b_n - a_n}, & x \in [a_n, b_n] \\ 0, & otherwise \end{cases}.$$

Since the intervals are independent, the joint density function (JDF) for the *n* possible execution time intervals is:

$$f(x_1, x_2, ..., x_n) = \prod_n f_n(x) = \begin{cases} \prod_n \dfrac{1}{b_n - a_n}, x_n \in [a_n, b_n] \forall x_n \\ 0, \quad otherwise \end{cases}$$

Accordingly, $f(x_n)$ evaluates to be a constant, c. The cumulative joint density function (CJDF) for n uniform distributions is:

$$F(x_1, x_2, ..., x_n) = \int_{y_1} \int_{y_2} \cdots \int_{y_n} c \; dx_1 dx_2 \cdots dx_n$$

From a practical standpoint, we evaluate $F(x_n)$ in separate intervals, which we shall call *evaluation intervals*. These intervals are the different intersections of the event intervals in the NOS. Figure 4 gives an example of the evaluation intervals for the events A, B and C. The evaluation intervals would be [1,2], [2,3] and [3,4] for this example.

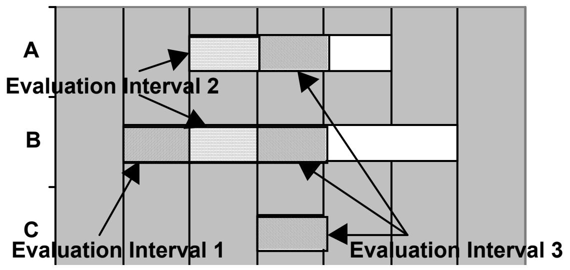

Figure 4: Example of evaluation intervals

Given this information, we can determine the score of each event using the probability that a given event would be executed first. As in the previous algorithm, let *S* represent the NOS of *N* events with overlapping intervals, i.e., $S = \{E_1, E_2, ..., E_N\}$. Additionally, let A_e (B_e) be the set of events scheduled by the execution of E_n whose intervals overlap with the remaining *N-1* events. The set A_e (B_e) may be empty if no such events are scheduled. In addition, let *T* be the current thread score.

The actual algorithm is as follows:

```
Procedure ComputeRank(N, S, T)
    For i = 1,2, ..., N
        Set $s_i = 0$.
    End For
    Set x = first evaluation interval
    While $x \neq \varnothing$
        Set l = lower boundary of evaluation interval
        Set u = upper boundary of evaluation interval
        ' Now loop through $2^n$ probability combinations
        for the n intervals. For two intervals, $X_1$ and $X_2$,
        the four probability combinations would be
        $X_1 \cap X_2$, $X_1 \cap \sim X_2$, $\sim X_1 \cap X_2$, $\sim X_1 \cap \sim X_2$.
        For i = 1, 2, ..., $2^N$
            ' Evaluate the cumulative joint density
            function for the probability combination. For
            $X_1 \cap X_2$, the CJDF evaluated would be
            $\int_l^u \int_l^u c \; dx_1 dx_2$ . For the probability
            combination $X_1 \cap \sim X_2$, the CJDF would be
            $\int_l^u \int_{max(u,a_2)}^{b_2} c \; dx_1 dx_2$ .
            Set p = CJDF of probability combination
            Set q = number of intervals in S that intersect
            [l,u]
            For j = 1, 2, ..., N
                If ($E_j \cap [l,u]$) then
                    Set $s_j = s_j + p/q$.
                End If
            End For
        End For
        Set x = next evaluation interval
    End While
    For n = 1, 2, ..., N
        Set $T' = T * s_n$
        Calculate $A_e$ and $B_e$
        If ($N - 1 + |A_e| - |B_e| \geq 2$) then
            $S = S + A_e - E_n - B_e$
            ComputeRank($N - 1 + |A_e| - |B_e|$ , S, T')
        End If
    End For
End Procedure ComputeRank
```

Now let us use Figure 2 as an example of how this algorithm would work. The JDF for these three intervals is 1/((5-2)*(6-1)*(4-3)) = 1/15. The first evaluation interval is [1,2]. The only non-zero CJDF in this interval is

~A∩B∩~C, and it would be evaluated as $\int_2^5\int_1^2\int_3^4 \frac{1}{15}\, da\, db\, dc = \frac{1}{5}$. So B would be given the score 1/5.

The next evaluation interval is [2,3]. The non-zero CJDFs in this interval are A∩B∩~C, A∩~B∩~C, and ~A∩B∩~C. They would be evaluated as $\int_2^3\int_2^3\int_3^4 \frac{1}{15}\, da\, db\, dc = \frac{1}{15}$, $\int_2^3\int_3^6\int_3^4 \frac{1}{15}\, da\, db\, dc = \frac{1}{5}$, and $\int_3^5\int_2^3\int_3^4 \frac{1}{15}\, da\, db\, dc = \frac{2}{15}$, respectively. The score for A would be (1/15)/2 + 1/5 = 7/30. The additional score for B would be (1/15)/2 + 2/15 = 1/6. The total score for B would be 1/6 + 1/5 = 11/30.

The last evaluation interval is [3,4]. It is the last evaluation interval because at least one of the three events must execute before time 4. The non-zero CJDFs in this interval are A∩B∩C, A∩~B∩C, ~A∩B∩C, and ~A∩~B∩C. They would be evaluated as $\int_3^4\int_3^4\int_3^4 \frac{1}{15}\, da\, db\, dc = \frac{1}{15}$, $\int_3^4\int_4^6\int_3^4 \frac{1}{15}\, da\, db\, dc = \frac{2}{15}$, $\int_4^5\int_3^4\int_3^4 \frac{1}{15}\, da\, db\, dc = \frac{1}{15}$,and $\int_4^5\int_4^6\int_3^4 \frac{1}{15}\, da\, db\, dc = \frac{2}{15}$, respectively. For A, the additional score would be (1/15)/3 + (2/15)/2 = 4/45. The total score for A would be 4/45 + 7/30 = 29/90 ≈ .3222. The additional score for B would be (1/15)/3 + (1/15)/2 = 1/18. The total score for B would be 1/18 + 11/30 = 19/45 ≈ .4222. The total score for C would be (1/15)/3 + (2/15)/2 + (1/15)/2 + 2/15 = 23/90 ≈ .2556.

Let us assume that A actually is the first event to be executed. The thread where A is first event to be executed would have a time of [2,4]. The current thread score would be .3222. The remaining events would be B and C. The JDF for comparing B and C for this thread would be 1/((6-2)*(4-3)) = 1/4.

In the first possible interval, [2,3], only B can be executed. The only non-zero CJDF in the interval [2,3] is B∩~C. The score for B would be evaluated as $\int_2^3\int_3^4 \frac{1}{4} db\, dc = \frac{1}{4}$. So B would be given the score 1/4.

The next evaluation interval would be [3,4]. The non-zero CJDF's in this interval are B∩C and ~B∩C. They would be evaluated as $\int_3^4\int_3^4 \frac{1}{4} db\, dc = \frac{1}{4}$, and $\int_4^6\int_3^4 \frac{1}{4} db\, dc = \frac{1}{2}$. The additional score for B would be (1/4)/2 = 1/8, making the total score for B = 3/8 = .375. The score for C would be (1/4)/2 + 1/2 = 5/8 = .625. Therefore, the total score for the sequence ABC would be .3222 * .375 * 1 = 0.1208. The total score for the sequence ACB would be .3222 * .625 = 0.2014.

The scoring of the other sequences shows the following results:

Sequence	Score
ABC	0.1208
ACB	0.2014
BAC	0.2111
BCA	0.2111
CAB	0.1704
CBA	0.0852

4 COMPARISONS USING PERT EXAMPLES

In Figures 5-8, we see four different PERT examples, each taken from Johnson and Montgomery (1974). These PERT networks are used to determine which of the qualitative scoring methodologies more closely aligns with randomly generated sequences of these networks. For each PERT network, we have run qualitative simulations using each of the scoring methods and run each of the PERT networks for 1000 iterations using uniformly distributed event times and triangular distributed event times. For the triangular distribution, the mode was set to the midpoint of the interval. For the comparisons, we have not included the Rank method because the Midpoint Multiple method is a more refined version of the Rank method.

The translation of a PERT diagram to an event graph is straightforward. Each vertex in the graph is an event. The vertex knows the number of incoming edges to that event. When the event is triggered, it counts the number of times that it has been "hit". When the number of hits is equal to the number of incoming edges, then event schedules the outgoing edges. This continues until there are no more events scheduled.

In Figures 9-11, we show output examples of a given thread in a simulation and how the methods would score that particular thread. The *Thread* header is the number of the thread in the simulation. The *Event* header is the event number in the simulation. The *Score* header shows the accumulated score for that thread scoring method. The *Clock* header is the current simulation time. The *Head* and *Tail* headers show the head and tail of the arc that was just executed. The *Future Events Calendar* shows the events that are scheduled to be executed. Each record in the future events calendar is configured as follows: The time the event is to be executed, the arc that is to be executed, the priority of the event in the calendar (lowest value has high priority) and the time that the event was scheduled.

4.1 The Midpoint Ranking Method

An example of the Midpoint Ranking Method is seen in Figure 9. This figure shows the first thread of PERT Example 1 (See Figure 5). In this example, the thread gets the top ranked event, except when event 6 is executed. At that point, the first event on the calendar has an execution time of [10,32] and a midpoint of 21. The second has an execution time of [11,29] and a midpoint of 20.5. As a result, because thread 1 executes the Arc(3,5) event, it is assigned a score of 2, which is added to the previous score of 6 for a total of 8.

4.2 The Midpoint Multiple Method

To score the same thread using the Midpoint Multiple Method, we would assign a 1 for every event except for event 7. For event 7, we have the same problem, in that Arc(4,6) has an earlier midpoint than Arc(3,5). Arc(4,6) is given a score of 1 and Arc(3,5) is given a score of (21 – 10)/(20 – 10) = 1.1. Again, when it is added to the previous score of 6, the total is 7.1. This sequence can be seen in Figure 10.

4.3 The Uniform Distribution Method

In order to give an example of the scoring method, It is obvious from the graph that at time [2,6] there will be three events on the calendar and that any one of those events could be executed first. Arc(2,5) is scheduled at time [5,15],

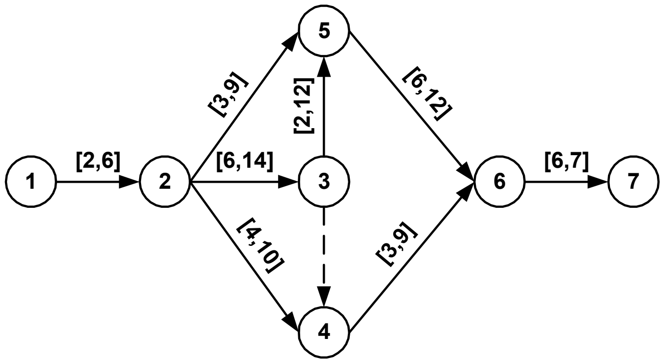

Figure 5: PERT example 1

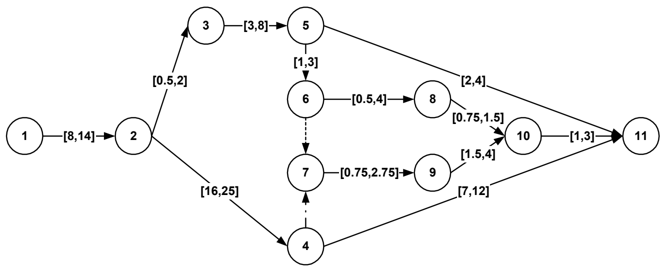

Figure 7: PERT example 3

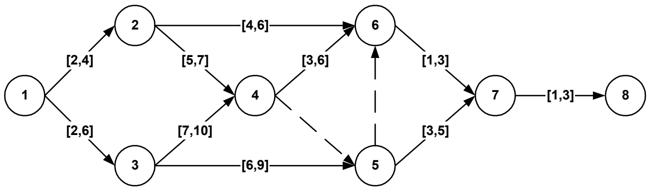

Figure 6: PERT example 2

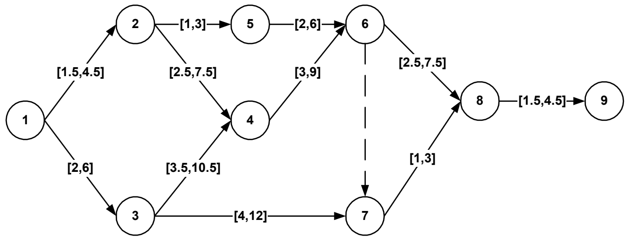

Figure 8: PERT example 4

						Future Events Calendar		
Thread	Event	Score	Clock	Head	Tail	Event 1	Event 2	Event 3
1	1	1	[0,0]	0	1	[2.0,6.0],Arc(1,2),3,[0,0]		
1	2	2	[2.0,6.0]	1	2	[5.0,15.0],Arc(2,5),3,[2.0,6.0]	[6.0,16.0],Arc(2,4),3,[2.0,6.0]	[8.0,20.0],Arc(2,3),3,[2.0,6.0]
1	3	3	[5.0,15.0]	2	5	[6.0,16.0],Arc(2,4),3,[2.0,6.0]	[8.0,20.0],Arc(2,3),3,[2.0,6.0]	
1	4	4	[6.0,16.0]	2	4	[8.0,20.0],Arc(2,3),3,[2.0,6.0]		
1	5	5	[8.0,20.0]	2	3	[8.0,20.0],Arc(3,4),2,[8.0,20.0]	[10.0,32.0],Arc(3,5),3,[8.0,20.0]	
1	6	6	[8.0,20.0]	3	4	[10.0,32.0],Arc(3,5),3,[8.0,20.0]	[11.0,29.0],Arc(4,6),3,[8.0,20.0]	
1	7	8	[10.0,29.0]	3	5	[11.0,29.0],Arc(4,6),3,[8.0,20.0]	[16.0,41.0],Arc(5,6),3,[10.0,29.0]	
1	8	9	[11.0,29.0]	4	6	[16.0,41.0],Arc(5,6),3,[10.0,29.0]		
1	9	10	[16.0,41.0]	5	6	[22.0,48.0],Arc(6,7),3,[16.0,41.0]		
1	10	11	[22.0,48.0]	6	7			

Figure 9: Midpoint ranking method scoring

						Future Events Calendar		
Thread	Event	Score	Clock	Head	Tail	Event 1	Event 2	Event 3
1	1	1	[0,0]	0	1	[2.0,6.0],Arc(1,2),3,[0,0]		
1	2	2	[2.0,6.0]	1	2	[5.0,15.0],Arc(2,5),3,[2.0,6.0]	[6.0,16.0],Arc(2,4),3,[2.0,6.0]	[8.0,20.0],Arc(2,3),3,[2.0,6.0]
1	3	3	[5.0,15.0]	2	5	[6.0,16.0],Arc(2,4),3,[2.0,6.0]	[8.0,20.0],Arc(2,3),3,[2.0,6.0]	
1	4	4	[6.0,16.0]	2	4	[8.0,20.0],Arc(2,3),3,[2.0,6.0]		
1	5	5	[8.0,20.0]	2	3	[8.0,20.0],Arc(3,4),2,[8.0,20.0]	[10.0,32.0],Arc(3,5),3,[8.0,20.0]	
1	6	6	[8.0,20.0]	3	4	[10.0,32.0],Arc(3,5),3,[8.0,20.0]	[11.0,29.0],Arc(4,6),3,[8.0,20.0]	
1	7	7.1	[10.0,29.0]	3	5	[11.0,29.0],Arc(4,6),3,[8.0,20.0]	[16.0,41.0],Arc(5,6),3,[10.0,29.0]	
1	8	8.1	[11.0,29.0]	4	6	[16.0,41.0],Arc(5,6),3,[10.0,29.0]		
1	9	9.1	[16.0,41.0]	5	6	[22.0,48.0],Arc(6,7),3,[16.0,41.0]		
1	10	10.1	[22.0,48.0]	6	7			

Figure 10: Midpoint multiple method scoring

						Future Events Calendar		
Thread	Event	Score	Clock	Head	Tail	Event 1	Event 2	Event 3
1	1	1	[0,0]	0	1	[2.0,6.0],Arc(1,2),3,[0,0]		
1	2	1	[2.0,6.0]	1	2	[5.0,15.0],Arc(2,5),3,[2.0,6.0]	[6.0,16.0],Arc(2,4),3,[2.0,6.0]	[8.0,20.0],Arc(2,3),3,[2.0,6.0]
1	3	0.5269	[5.0,15.0]	2	5	[6.0,16.0],Arc(2,4),3,[2.0,6.0]	[8.0,20.0],Arc(2,3),3,[2.0,6.0]	
1	4	0.3864	[6.0,16.0]	2	4	[8.0,20.0],Arc(2,3),3,[2.0,6.0]		
1	5	0.3864	[8.0,20.0]	2	3	[8.0,20.0],Arc(3,4),2,[8.0,20.0]	[10.0,32.0],Arc(3,5),3,[8.0,20.0]	
1	6	0.3864	[8.0,20.0]	3	4	[10.0,32.0],Arc(3,5),3,[8.0,20.0]	[11.0,29.0],Arc(4,6),3,[8.0,20.0]	
1	7	0.1756	[10.0,29.0]	3	5	[11.0,29.0],Arc(4,6),3,[8.0,20.0]	[16.0,41.0],Arc(5,6),3,[10.0,29.0]	
1	8	0.1427	[11.0,29.0]	4	6	[16.0,41.0],Arc(5,6),3,[10.0,29.0]		
1	9	0.1427	[16.0,41.0]	5	6	[22.0,48.0],Arc(6,7),3,[16.0,41.0]		
1	10	0.1427	[22.0,48.0]	6	7			

Figure 11: Uniform distribution method scoring

Arc(2,4) is schedule at time [6,16] and Arc(2,3) is scheduled at time [8,20]. Using the uniform distribution method, the scores are as follows:

Arc(2,5) = 0.5269444
Arc(2,4) = 0.3573611
Arc(2,3) = 0.1156945

Overall, for thread 1, the score is 0.1427, as can be seen in Figure 11. Figure 11 shows each of the events, the scores assigned to the events, and the running score for the thread.

4.4 Comparisons against Random PERT Networks

In an effort to determine if these ranking algorithms correlate to random experiments on the same networks, we took the four PERT networks in Figures 5-8 and conducted the following experiments:

1. We ran 1000 replications using randomly generated uniformly distributed activity times and ranked threads (event sequences) by the number of occurrences.
2. We ran another 1000 replications using randomly generated triangular distributions with the mode being the midpoint of the interval. Again, we ranked the threads by the number of occurrences.

		TOP 1				TOP 1	
		Midpoint				Midpoint	
% Agreement	Threads	Multiple	Uniform	Lowest Rank in	Threads	Multiple	Uniform
Example 1	37	100%	100%	Example 1	37	1	1
Example 2	669	100%	100%	Example 2	669	1	1
Example 3	351	100%	0%	Example 3	351	1	2
Example 4	168	100%	100%	Example 4	168	1	1
		TOP 3				TOP 3	
		Midpoint				Midpoint	
		Multiple	Uniform			Multiple	Uniform
Example 1		100%	100%	Example 1		3	3
Example 2		67%	67%	Example 2		7	5
Example 3		100%	67%	Example 3		3	8
Example 4		67%	67%	Example 4		11	5
		TOP 5				TOP 5	
		Midpoint				Midpoint	
		Multiple	Uniform			Multiple	Uniform
Example 1		100%	100%	Example 1		5	5
Example 2		40%	60%	Example 2		13	10
Example 3		N/A	N/A	Example 3		N/A	N/A
Example 4		80%	80%	Example 4		11	6
		TOP 10				TOP 10	
		Midpoint				Midpoint	
		Multiple	Uniform			Multiple	Uniform
Example 1		N/A	N/A	Example 1		N/A	N/A
Example 2		70%	80%	Example 2		15	13
Example 3		N/A	N/A	Example 3		N/A	N/A
Example 4		60%	70%	Example 4		23	17

Figure 12: Random uniform distribution rankings vs. qualitative thread rankings

		TOP 1				TOP 1	
		Midpoint				Midpoint	
% Agreement	Threads	Multiple	Uniform	Lowest Rank in	Threads	Multiple	Uniform
Example 1	37	100%	100%	Example 1	37	1	1
Example 2	669	100%	100%	Example 2	669	1	1
Example 3	351	100%	0%	Example 3	351	1	2
Example 4	168	100%	100%	Example 4	168	1	1
		TOP 3				TOP 3	
		Midpoint				Midpoint	
		Multiple	Uniform			Multiple	Uniform
Example 1		100%	100%	Example 1		3	3
Example 2		67%	67%	Example 2		7	5
Example 3		100%	67%	Example 3		3	8
Example 4		67%	67%	Example 4		11	5
		TOP 5				TOP 5	
		Midpoint				Midpoint	
		Multiple	Uniform			Multiple	Uniform
Example 1		80%	80%	Example 1		6	6
Example 2		80%	100%	Example 2		7	5
Example 3		N/A	N/A	Example 3		N/A	N/A
Example 4		80%	80%	Example 4		11	6
		TOP 10				TOP 10	
		Midpoint				Midpoint	
		Multiple	Uniform			Multiple	Uniform
Example 1		N/A	N/A	Example 1		N/A	N/A
Example 2		70%	80%	Example 2		13	12
Example 3		N/A	N/A	Example 3		N/A	N/A
Example 4		60%	70%	Example 4		23	17

Figure 13: Random triangular distribution rankings vs. qualitative thread rankings

After these experiments were run, we determined how the Midpoint Multiple and the Uniform ranking methods for the qualitative networks compared to these two randomly generated ranking. When we refer to "Top 5", we are referring to the top 5 threads ranked by these random experiments.

Figures 12 and 13 show how the Midpoint Multiple and the Uniform Distribution scoring methods compare against threads generated by the Random Uniform experiment and the Random Triangular Experiment, respectively. On the left-hand side of each figure, we show how often the qualitative ranking agrees with the random ranking on a percentage basis. A 100% agreement means that the qualitative ranking ranked the same Top 5 threads, for example, as the random experiment. However, it does not mean that the thread rankings were identical. The right-hand side shows the lowest ranked thread of the qualitative ranking method that was ranked in the Top 5, for example, of the random experiment.

In the Top 5 of Figure 12, the two qualitative ranking methods agreed with the random experiment of Example 4 on 80% of the threads. The only thread that did not agree for the Midpoint Multiple method was the thread that was ranked 11th by the Midpoint Multiple Method. The only thread that did not agree for the Uniform Distribution method was ranked 6th by the Uniform Distribution method.

In Figure 12, we see general agreement between the qualitative ranking methods and the randomly generated experiment. The Uniform Distribution method did agree more often than the Midpoint Multiple method. When the Uniform Distribution method did not agree, then the miss was not as severe as the Midpoint Multiple method. The same can be said for Figure 13.

Overall, the rankings for both qualitative methods show some issues, but considered the number of threads that were ranked; the "missed" rankings are a very small percentile of the overall number of threads. Also, it looks as if the qualitative ranking methods better agree with the Triangular Distribution experiment. This may mean that the qualitative ranking methods are more favorable with modal distribution, as compared to random experiments using the uniform distribution.

5 CONCLUSIONS

The three qualitative thread ranking methods, Midpoint Rank, Midpoint Multiple Rank, and Uniform Distributed Rank all show relatively good thread ranking vs. some randomly generated PERT networks. The Midpoint Rank method has an issue because it does not have differentiation capability among many different threads, so we consider that method not to be good in practical use. However, both the Midpoint Multiple and the Uniform have ways of better differentiating between threads and both seem to track well with randomly generated PERT networks. When compared with the other qualitative ranking methods presented here, the Uniform Distribution method yields slightly better results when compared to randomly generated rankings.

One aspect of thread ranking is that we could use the qualitative thread ranking to rank the least likely threads. These threads could be unlikely, but have negative consequences it they were to occur. We did not address how these qualitative ranking methods actually predict these least likely threads.

ACKNOWLEDGMENTS

We gratefully acknowledge the financial support of the CBA/GSB Faculty Research Committee of the College of Business Administration of the University of Texas at Austin.

REFERENCES

Forbus, K. D. 1988. Qualitative Physics: Past, Present, and Future. *Exploring Artificial Intelligence*. Howard Shrobe, ed. San Mateo: Morgan Kaufmann Publishers, Inc.

Hinkkanen, A., K. R. Lang, and A. B. Whinston. 1993. A Theoretical Foundation of Qualitative Reasoning Based on Set Theory. Working Paper.

Ingalls, R. G., D. J. Morrice, A. B. Whinston. 2000. The Implementation of Temporal Intervals in Qualitative Simulation Graphs. *ACM Transactions on Modeling and Computer Simulation* 10(3):215-240.

Ingalls, R. G., D. J. Morrice, E. Yücesan, A. B. Whinston. 2003. Execution Conditions: a Formalization of Event Cancellation in Simulation Graphs. *INFORMS Journal on Computing* 15(4):397-411.

Johnson, L. A. and D. C. Montgomery. 1974. *Operations Research in Production Planning, Scheduling, and Inventory Control.* New York, New York: John Wiley and Sons, Inc.

Schruben, L. W. 1983. Simulation Modeling with Event Graphs. *Communications of the ACM* 26(11):957-963.

Som, T. K. and R. G. Sargent. 1989. Formal Development of Event Graphs as an Aid to Structured and Efficient Simulation Programs. *ORSA Journal on Computing* 1(2):107-125.

Yücesan, E. 1990. *Simulation Graphs: A Mathematical Framework for the Design and Analysis of Discrete Event Simulations*. Ph.D. Dissertation, School of Operations Research and Industrial Engineering, Ithaca, New York.

AUTHOR BIOGRAPHIES

RICKI G. INGALLS is Associate Professor and Site Director of the Center for Engineering Logistics and Distribution (CELDi) in the School of Industrial Engineering and Management at Oklahoma State University. He has developed a graduate program in Supply Chain Engineering where he teaches Supply Chain Strategy and Supply Chain Modeling. Through CELDi and an active consulting practice, Dr. Ingalls has consulted with companies and government agencies in areas such as business strategy, business modeling, resource planning, labor planning, and freight modeling. Dr. Ingalls joined Oklahoma State in 2000 after 16 years in industry with Compaq, SEMATECH, General Electric and Motorola. His last position at Compaq was an executive position reporting to the Vice-President of Global Integrated Logistics where he was responsible for Supply Chain Design projects and Supply Chain Strategic Planning for the corporation. He has a B.S. in Mathematics from East Texas Baptist College (1982), a M.S. in Industrial Engineering from Texas A&M University (1984) and a Ph.D. in Management Science from the University of Texas at Austin (1999). His email address is <ricki.ingalls@okstate.edu>.

DOUGLAS J. MORRICE is a Professor in Operations Management at The University of Texas at Austin. He has ORIE Ph.D. from Cornell University. His research interests include simulation design, modeling, and analysis. Dr. Morrice was Co-Editor of the *Proceedings of the 1996 Winter Simulation Conference*, and 2003 Winter Simulation Conference Program Chair. He is currently serving as a representative for the INFORMS Simulation Society on the Winter Simulation Conference Board of Directors. His email address is <morrice@mail.utexas.edu >.