REVUE FRANÇAISE D'AUTOMATIQUE, D'INFORMATIQUE ET DE RECHERCHE OPÉRATIONNELLE. RECHERCHE OPÉRATIONNELLE

SHELDON H. JACOBSON
ENVER YÜCESAN

Intractability results in discrete-event simulation

Revue française d'automatique, d'informatique et de recherche opérationnelle. Recherche opérationnelle, tome 29, n° 3 (1995), p. 353-369.

<http://www.numdam.org/item?id=RO_1995__29_3_353_0>

Recherche opérationnelle/Operations Research
(vol. 29, n° 3, 1995, p. 353 à 369)

INTRACTABILITY RESULTS IN DISCRETE-EVENT SIMULATION (*)

by Sheldon H. JACOBSON (1) and Enver YÜCESAN (2)

Abstract. – *Simulation is often viewed as a modeling methodology of last resort. This is due to the lack of automated algorithms and procedures that exist to aid in the construction and analysis of simulation models. Jacobson and Yücesan (1994) present four structural issue search problems associated with simulation model building, and prove them to be* NP-hard, *hence intractable under the worst-case analysis of computational complexity theory. In this article, three new structural issue search problems are presented and proven to be* NP-hard. *The consequences and implications of these results are discussed.*

Keywords: Simulation: Model Building, Computational Complexity, Discrete Event Systems.

Résumé. – *La simulation est souvent considérée comme le dernier outil de modélisation et d'analyse. Ceci est en raison de l'absence des algorithmes et des procédures qui peuvent soutenir le développement et l'analyse des modèles de simulation. Jacobson et Yücesan (1994) présentent quatre problèmes associés à la modélisation et prouvent qu'ils sont* NP-hard *au sein de la théorie de la complexité de computation. Dans cet article, trois nouveaux problèmes sont présentés et sont prouvés* NP-hard. *Les conséquences de ces résultats sont également considérées.*

Mots clés : Simulation: Modélisation, Complexité de computation, Systèmes à Evénements Discrets.

1. INTRODUCTION

Discrete event simulation models are used to analyze complex, large-scale, real-world systems. A severe limitation of this modeling approach is the lack of automated tools to help practitioners construct valid models. In addition, once such models are built, there are no automated tools available to efficiently analyze their structural properties. The best one can hope to have

The first author was supported in part by the National Science Foundation (DMI-9409266) and the Air Force Office of Scientific Research (94NM378).

(*) Received December 1993.

(1) Department of Industrial and Systems Engineering, Virginia Polytechnic Institute and State University, Blacksburg, Virginia 24061-0118, U.S.A.

(2) INSEAD, European Institute of Business Administration, Boulevard de Constance, 77305 Fontainebleau Cedex, France.

Recherche opérationnelle/Operations Research, 0399-0559/95/03/$ 4.00

are model-specific techniques and procedures with little or no guidelines on when these methods are valid.

Jacobson and Yücesan (1994) define four search problems for discrete event models, and prove them to be NP-hard, hence intractable under the worst case analysis of computational complexity theory, unless P=NP (Garey and Johnson, 1979). These four problems are accessibility, ordering, noninterchangeability, and stalling. Informally, *accessibility* asks whether it is possible to find a sequence of events in a simulation model such that, when it is executed, a particular state is reached. *Ordering* asks whether it is possible to find a sequence of events in a simulation model such that, when executed with the order of the last two events interchanged, two distinct states are reached. *Noninterchangeability* asks whether it is possible to find a sequence of events in a simulation model such that when it is executed in two separate implementations of the simulation model (e.g., the implementation of the same model using two different simulation languages), two distinct states are reached. *Stalling* asks whether it is possible to find a sequence of events in a simulation model such that, when it is executed, a particular state is reached where the future events list is empty and the stopping condition is not satisfied.

There are several theoretical and practical implications of these four search problems being NP-hard. On the theoretical side, these results provide a unifying framework to explain the difficulty of seemingly different simulation modeling and analysis issues. It also explains why simulation researchers have been unable to automate simulation model building and analysis tasks, and why there is a need for the many simulation languages and simulators on the market (no one package can satisfy all the requirements of practitioners). On the practical side, diverse issues such as model validation and verification, guaranteed variance reduction, and validation of the applicability of sample path based techniques are all impacted by the NP-hardness results in Jacobson and Yücesan (1994). For an extensive discussion of these implications, *see* Yücesan and Jacobson (1992).

In this paper, three new search problems associated with structural issues in simulation modeling are defined and proven to be NP-hard. Their implications are discussed not only for simulation model building and analysis but also for performance assessment of general discrete-event dynamic systems (DEDS). The paper is organized as follows: in Section 2, formal definitions are presented that are needed to obtain the main results. Three new search problems are also presented and proven to be NP-hard. Section 3 discusses

the implications of these results, while Section 4 summarizes the highlights of the paper.

2. THREE SIMULATION SEARCH PROBLEMS AND THEIR COMPLEXITY

Three simulation model structural issue search problems are presented. The first problem focuses on finding a sequence of events such that executing the sequence in any valid order or permutation results in different states being reached. The second problem focuses on finding a sequence of events such that executing the sequence in any valid order or permutation results in different future events lists being produced. The third problem focuses on finding a sequence of events such that executing the sequence results in future events being cancelled.

2.1 Definitions

To establish the complexity results, the concepts associated with a simulation model must be defined. These definitions are taken from Jacobson and Yücesan (1994).

A *model specification* is a representation of the system under study, reflecting the objectives of the study and the assumptions of the analysis. A model specification can be in the form of a Generalized Semi-Markov Process (GSMP) (*see* Glasserman and Yao, 1992a) or a Simulation Graph (*see* Schruben and Yücesan, 1993). The *state* of a system is a value that provides a complete description of the system, including values for all of its attributes as well as any schedule for the future. *Events* induce changes in the state of the system. There are a countable number of event types. A *model implementation* is a translation of the model specification into a computer executable form. This could be in a high-level programming language or in a particular simulation package. Informally, a model specification defines what a model does while the model implementation defines how the model behavior is to be achieved. An event sequence is said to be *valid* if the sequence is well defined in a simulation model specification.

The theory of *computational complexity* provides a well-defined framework to assess the tractability of decision problems (Garey and Johnson, 1979). A *decision problem* is one whose solution is either "yes" or "no". In general, we are interested in finding the most "efficient" algorithm to solve a problem. Typically, the fastest algorithm is considered as the most efficient. We then define the *time complexity function* for an algorithm as the maximum amount of time needed by the algorithm to solve a problem instance of a particular size, which represents a worst-case performance criterion.

Note that the *size* of a problem instance is defined as the amount of input data needed to describe the instance. Various encoding schemes are possible to describe a problem instance. The most widely accepted scheme, which is the one adopted here, is the number of tape cells on a Turing machine.

A *polynomial-time algorithm* is one whose time complexity function is $O(p(n))$, where p is a polynomial function and n denotes the size of the problem instance. Any algorithm whose time complexity function cannot be so bounded is called an *exponential-time algorithm.* Given the explosive growth rates for exponential complexity functions, polynomial-time algorithms are much more desirable from a practical point of view. It is well accepted that a problem is not "well-solved" until a polynomial-time algorithm has been found for it (Garey and Johnson, 1979; p. 8).

Decision problems in the class NP are those problems for which a potential solution can be verified in polynomial time in the size of the problem instance. The complete problems for this class (that is, NP-complete problems) are the hardest problems in NP such that, if one such problem could be solved in polynomial time, then all problems in NP could be solved in polynomial time. Moreover, NP-hard problems are search problems which are provably at least as hard as NP-complete decision problems.

We assume that the size of a discrete event simulation model specification is n. This is the number of tape cells on a Turing machine required to represent a model implementation of the model specification such that it can be executed. Note that model implementations are not unique, in that there are several possible model implementations associated with each model specification. Any event of the model specification is also assumed to be executable in polynomial time in n; that is, $p_1(n)$. This assumption restricts our work to a subclass of simulation models. Such a subclass, however, contains all of the relevant simulation models from a practical point of view, as models whose events could take an exponential amount of time to execute would not have much use in a simulation study. These assumptions will be used in all the subsequent theorems, unless it is otherwise stated.

The notation $E_0\, E_1\, E_2 \ldots E_k \rightarrow S$ denotes that the event sequence, when executed, leads to the state S, while the notation $E_0\, E_1\, E_2 \ldots E_k \nrightarrow S$ means that the event sequence, when executed, leads to any state except S. The initial event, E_0, establishes the initial state of the system and schedules further events to initiate the execution of the simulation model implementation.

2.2 Search Problems and Results

Consider the following structural problem, termed STRONG PERMUTATION. Informally, STRONG PERMUTATION asks whether a sequence of events and a one-to-one (valid) permutation of these events can be found such that when the sequence of events and the permutation of the events are executed, both starting from the same initial event, different states are reached.

STRONG PERMUTATION

Instance: – A discrete event simulation model specification with an associated simulation model implementation,
– An initial event, E_0,
– Two distinct states, S_1 and S_2,
– A non-negative finite integer, K.

Question: Find a sequence of events $E_1, E_2, \ldots, E_k$, with $k \leq K$, and a one-to-one permutation function

$$\pi : \{1, 2, , \ldots k\} \rightarrow \{1, 2, \ldots, k\}$$

such that

$$E_0\, E_1\, E_2 \ldots E_k \rightarrow S_1$$

and

$$E_0\, E_{\pi(1)}\, E_{\pi(2)} \ldots E_{\pi(k)} \rightarrow S_2,$$

where $E_0\, E_{\pi(1)}\, E_{\pi(2)} \ldots E_{\pi(k)}$ is a valid event sequence.

The following theorem proves STRONG PERMUTATION to be NP-hard.

THEOREM 1: STRONG PERMUTATION *is* NP-hard.

Proof: To show that STRONG PERMUTATION is NP-hard, a polynomial Turing reduction from ACCESSIBILITY to STRONG PERMUTATION will be constructed. First, ACCESSIBILITY is formally defined.

ACCESSIBILITY (Jacobson and Yücesan, 1994)

Instance: – A discrete event simulation model specification with an associated simulation model implementation,
– An initial event, E_0,

– A state, S,

– A non-negative finite integer, M.

Question: Find a sequence of events $E_1, E_2, \ldots, E_m$, with $m \leq M$, such that the execution of the sequence yields $E_0 E_1 E_2 \ldots E_m \rightarrow S$.

For a general instance of ACCESSIBILITY, define the associated particular instance of STRONG PERMUTATION as follows:

The discrete event simulation model specification and the associated simulation model implementation are the same as for ACCESSIBILITY with the following modifications: one additional state S_Z and one additional event F are defined. The initial event E_0 is defined in the same way as for ACCESSIBILITY except for one additional feature: it schedules event F with time delay t. Define state $S_2 = S$ from ACCESSIBILITY, and state $S_1 = S_Z$, a new state reachable only through the new event F, defined as:

$$F = \{\text{If STATE} = S_2, \text{ set STATE} \leftarrow S_Z \text{ and continue.}\}$$

Lastly, $K = M + 1$.

This reduction can be made in polynomial time in the size of the instance of ACCESSIBILITY. To complete the proof, it is necessary to show that a solution to STRONG PERMUTATION can be used to solve ACCESSIBILITY. Suppose that a solution to STRONG PERMUTATION can be found. Then there exists a sequence of $k \leq K$ events and a one-to-one permutation function π such that

$$E_0\, E_1\, E_2 \ldots E_k \rightarrow S_1 \quad \text{and} \quad E_0\, E_{\pi(1)}\, E_{\pi(2)} \ldots E_{\pi(k)} \rightarrow S_2.$$

Suppose F is not one of the k events, $E_1, E_2, \ldots, E_k$. This is impossible since $S_1 = S_Z$ can only be achieved through event F. Therefore, F must be executed at least once in the sequence. By definition, however, F can be scheduled at most once in the sequence. Thus, F must be executed exactly once.

Event F can be executed with STATE $= S$ or with STATE $\neq S$. The latter case is impossible since then state S_Z cannot be reached. Therefore, event F must be executed with STATE $= S$, that is, $E_0 E_1 E_2 \ldots E_j \rightarrow S$, $j \leq k-1$. The resulting subsequence of events solves ACCESSIBILITY. □

The following example illustrates the search problem STRONG PERMUTATION.

Example 1: Consider the simulation model specification for a single-server queueing system. This specification is fully defined by the events

INITIALIZE, BEGIN, and COMPLETE, corresponding to initializing the system, customer arrivals, and service completions, respectively. For simplicity of notation, label these events I, B, and C, respectively. Define the state Q to be the number of customers in the system. Set $K = 7$ and suppose E_0 initializes $Q = 0$. For this system, every valid permutation of events results in the same state being reached, hence a solution to STRONG PERMUTATION cannot be found. For example, the event sequence $IBBBBBCC$ can be permuted into 21 different event sequences, of which 14 are valid. It is easy to verify that all 14 of these valid event sequence permutations result in state $Q = 3$ being reached.

Suppose that the single-server queueing system is embellished with a restriction on the system size, namely, a capacity of two, such that customers are denied access to the system if they find, upon arrival, one customer already in service and another in the queue. For this model specification, the event sequence $IBCBBCB$ results in state $(Q = 2)$, while the permutation of this event sequence $IBBBCBC$ results in state $(Q = 1)$. For this model specification, the state Q is a function of the event sequence order, hence a solution to STRONG PERMUTATION exists for $S_1 = 2$, $S_2 = 1$, and $K \geq 6$. □

Consider the following problem, termed WEAK PERMUTATION. Informally, this problem seeks to find a sequence of events and a valid one-to-one permutation of these events such that when the sequence of events and the (valid) permutation of the events are executed, both starting with the same initial event, different future events lists result.

WEAK PERMUTATION

Instance: – A discrete event simulation model specification with an associated simulation model implementation,
– An initial event, E_0,
– Two distinct future events lists, $\mathcal{A}_1$ and $\mathcal{A}_2$,
– A non-negative finite integer, M.

Question: Find a sequence of events $E_1, E_2, \ldots, E_m$, with $m \leq M$, and a one-to-one permutation function

$$\pi : \{1, 2, \ldots, m\} \rightarrow \{1, 2, \ldots, m\}$$

such that

$$A(E_0\, E_1\, E_2 \ldots E_m) = \mathcal{A}_1$$

and

$$A(E_0\, E_{\pi(1)}\, E_{\pi(2)} \ldots E_{\pi(m)}) = \mathcal{A}_2$$

where $E_0\, E_{\pi(1)}\, E_{(2)} \ldots E_{\pi(m)}$ is a valid event sequence and $A(E_0\, E_1\, E_2 \ldots E_m)$ is defined as the set of events scheduled to occur after the event sequence $E_0\, E_1\, E_2 \ldots E_m$ has been executed (*i.e.*, the resulting future events list).

This problem is termed "weak", in contrast to the previous search problem (termed "strong") because each state uniquely defines an associated future events list. However, the converse is not necessarily true.

The following theorem proves WEAK PERMUTATION to be NP-hard.

THEOREM 2: WEAK PERMUTATION *is* NP-hard.

Proof: A polynomial Turing reduction from STRONG PERMUTATION to WEAK PERMUTATION will be constructed.

For a general instance of STRONG PERMUTATION, define the associated particular instance of WEAK PERMUTATION as follows: the model specification and model implementation are the same as for STRONG PERMUTATION, except for the definition of three additional events, F, G, and H, where

F = {If STATE = S_1, then cancel all events on the events list and schedule event G with the highest priority.

If STATE = S_2, then cancel all events on the events list and schedule event H with the highest priority.

Else cancel all events on the events list and terminate the simulation}.

G = {Terminate the simulation}.

H = {Terminate the simulation}.

Note that events F, G, and H are uniquely defined for WEAK PERMUTATION and are not part of STRONG PERMUTATION. F is initially scheduled by E_0 with a time delay t. Also define $\mathcal{A}_1 = \{G\}$ and $\mathcal{A}_2 = \{H\}$. Lastly, set $M = K + 1$.

This reduction can be made in polynomial time in the instance of STRONG PERMUTATION. To complete the proof, it is necessary to show that a solution to WEAK PERMUTATION can be used to construct a solution to STRONG PERMUTATION. Suppose a solution to WEAK PERMUTATION exists. Therefore, there exists a sequence of M or fewer

events and a permutation function π such that $A(E_0\, E_1\, E_2 \ldots E_m) = \{G\}$ and $A(E_0\, E_{\pi(1)}\, E_{\pi(2)} \ldots E_{\pi(m)}) = \{H\}$. To establish that this can be used to construct a solution to STRONG PERMUTATION, the following cases must be considered:

(i) Event F can be executed at most once. This follows by the fact that event F can only be scheduled by event E_0.

(ii) Event F must be executed exactly once. This follows since event G or event H are on the resulting future events lists, and events G and H can only be scheduled by event F.

(iii) It is impossible for either event G or event H to be part of the solution sequence. This follows by the definition of event F, which can schedule events G or H at most once.

(iv) It is impossible for event F to be E_i, $1 \leq i \leq m-1$, nor $E_{\pi(j)}$, $1 \leq j \leq m-1$. To see this, suppose that event F were one of these events in the resulting event sequence. If the state is not equal to S_1 or S_2 when event F is about to be executed, then, by the definition of F, it is not possible to achieve the given future events lists. If the state is equal to S_1 (S_2) when F is about to be executed, then G (H) must be executed immediately, which contradicts (iii).

Therefore, events E_m and $E_{\pi(m)}$ must both be event F. In addition, since

$$A(E_0\, E_1\, E_2 \ldots E_{m-1}\, F) = \{G\}$$

and

$$A(E_0\, E_{\pi(1)}\, E_{\pi(2)} \ldots E_{\pi(2)} \ldots E_{\pi(m-1)}\, F) = \{H\},$$

then $E_0\, E_1\, E_2 \ldots E_{m-1} \rightarrow S_1$ and $E_0\, E_{\pi(1)}\, E_{\pi(2)} \ldots E_{\pi(m-1)} \rightarrow S_2$, which provide a solution to STRONG PERMUTATION. □

The following example illustrates the search problem WEAK PERMUTATION.

Example 2: Consider the embellished simulation model specification for the single-server queueing system of Example 1. For this model specification, the event sequence $IBCBBC$ results in $A(IBCBBC) = \{B, C\}$, while the permutation of this event sequence $IBBBCC$ results in $A(IBBBCC) = \{B\}$. For this model specification, the resulting future events list is a function of the event sequence order; hence, this event sequence solves WEAK PERMUTATION for $\mathcal{A}_1 = \{B, C\}$, $\mathcal{A}_2 = \{B\}$, and $M \geq 5$. □

Consider the following structural problem, termed INTERRUPT. Informally, INTERRUPT seeks a sequence of events such that events on the

future events list can be cancelled at some point during the execution of this event sequence. In other words, this condition requires that some pending event be interrupted before it is executed.

INTERRUPT

Instance: – A discrete event simulation model specification with an associated simulation model implementation,
– An initial event, E_0,
– A non-negative finite integer, K.

Question: Find a sequence of events $E_1, E_2, \ldots, E_k, E_{k+1}$, with $k \leq K$ and an event F such that

$$F \in A(E_0\, E_1\, E_2 \ldots E_k) \backslash E_{k+1}$$

while

$$F \notin A(E_0\, E_1\, E_2 \ldots E_k\, E_{k+1}).$$

The following theorem proves INTERRUPT to be NP-hard. The proof establishes a polynomial Turing reduction from 3-SATISFIABILITY (3-SAT) to INTERRUPT. First, 3-SAT is formally defined.

3-SATISFIABILITY (Garey and Johnson 1979, p. 46)

Instance: – A collection of clauses $V = \{V_1, V_2, \ldots, V_r\}$ on a finite set of Boolean variables $U = \{u_1, u_2, \ldots, u_t\}$ such that $|V_i| = 3$ for $i = 1, 2, \ldots, r$.

Question: – Does there exist a Boolean assignment for

$$U = \{u_1, u_2, \ldots, u_t\}$$

that satisfies all the clauses in V?

THEOREM 3: INTERRUPT *is* NP-hard.

Proof: A polynomial Turing reduction from 3-SAT to INTERRUPT will be constructed. The reduction is constructive, in that it creates a one-to-one mapping between the 3-SAT Boolean variable assignments that solve 3-SAT and the inputs, hence the sequence of events, to the discrete event simulation model specification instance of INTERRUPT.

Let $U = \{u_1, u_2, \ldots, u_t\}$ be a set of Boolean variables and let $V = \{V_1, V_2, \ldots, V_r\}$ be a set of clauses making up an arbitrary instance of 3-SAT. Without loss of generality, assume that a true (false) clause is represented by a 1 (0). A discrete event simulation model specification can be constructed in polynomial time in tr, whose implementation results in a solution to INTERRUPT that can be used to identify a Boolean assignment for U that satisfies all the clauses in V. The input to the simulation model specification consists of values for the a_j^i variables, deterministically set by the simulation user. These values are delay times in scheduling future events. Simultaneously scheduled events are executed using a first-in-first-out (FIFO) priority rule. Each event assigns a value of zero or one to Boolean variable. After all the Boolean variables are assigned a value, if all the clauses for the instance of 3-SAT are true, then an event on the future events list is cancelled, hence solving the instance of INTERRUPT constructed from the arbitrary instance of 3-SAT. In particular, the simulation is designed such that the solution to INTERRUPT yields a solution to 3-SAT.

Formally define the events for the discrete event simulation model specification as follows:

E_0 (Initialization):

Read values of input variables

$a_j^i \leftarrow 0$ or 1, $i = 1, 2, \ldots, t$, $j = 1, 2, 3, 4$,

$a_1^1 + a_2^1 = 1$ and $a_1^i + a_2^i + a_3^i + a_4^i = 3$, $i = 2, 3, \ldots, t$,

with the following constraints:

if $a_1^1 = 0$, then $a_1^2 + a_2^2 = 1$, $a_3^2 = a_4^2 = 1$,

if $a_2^1 = 0$, then $a_3^2 + a_4^2 = 1$, $a_1^2 = a_2^2 = 1$,

and for $j = 2, \ldots, t - 1$,

if $a_1^j = 0$ or $a_3^j = 0$, then $a_1^{j+1} + a_2^{j+1} = 1$, $a_3^{j+1} = a_4^{j+1} = 1$,

if $a_2^j = 0$ or $a_4^j = 0$, then $a_3^{j+1} + a_4^{j+1} = 1$, $a_1^{j+1} = a_2^{j+1} = 1$.

Set STATE $\leftarrow -1$.

Schedule E_1 with a time delay of a_1^1.

Schedule E_2 with a time delay of a_2^1.

For $k = 1, \ldots, t - 1$.

E_{2k-1} (Assign the value 0 to Boolean variable u_k):

Set $u_k \leftarrow 0$.

Schedule E_{2k+1} with a time delay of a_1^{k+1}.

Schedule E_{2k+2} with a time delay of a_2^{k+1}.

E_{2k} (Assign the value 1 to Boolean variable u_k):

Set $u_k \leftarrow 1$.

Schedule E_{2k+1} with a time delay of a_3^{k+1}.

Schedule E_{2k+2} with a time delay of a_4^{k+1}.

E_{2t-1} (Assign the value 0 to Boolean variable u_t):

Set $u_t \leftarrow 0$.

Schedule E_{2t+1} with no time delay.

E_{2t} (Assign the value 1 to Boolean variable u_t):

Set $u_t \leftarrow 1$.

Schedule E_{2t+1} with no time delay.

E_{2t+1} (Check the number of clauses to be true):

Set STATE $\leftarrow \sum_{j=1}^{r} V_j$.

Schedule E_{2t+2} with no time delay.

Schedule E_{2t+3} with no time delay, but with lower execution priority than E_{2t+2}.

E_{2t+2} (Cancel events):

If STATE $= r$, cancel E_{2t+3} and terminate the run.

If STATE $\neq r$, continue.

E_{2t+3} (Terminate):

Terminate the run.

The simulation model implementation associated with the simulation model specification can be in Sigma (Schruben, 1992). Lastly, $K = t + 2$. This discrete event simulation model specification is constructed in polynomial time in the size of the instance of 3-SAT. Suppose a solution to INTERRUPT is found. Then there exists a sequence of events such that the future events list has an event cancelled at some point in time. By the definition of the simulation model specification, this can only occur at event E_{2t+2} with

STATE $= r$. However, if STATE $= r$, then the answer to the instance of 3-SAT is yes. □

Note that a nonconstructive proof of Theorem 3 can be formulated by compressing events $E_1, E_2, \ldots, E_{2t}$ into a single event. However, the resulting simulation model specification will no longer yield a one-to-one mapping between the Boolean variable assignment that solves 3-SAT and the event sequence that solves INTERRUPT.

The following example illustrates the search problem INTERRUPT.

Example 3: Consider the simulation model specification for the single-server queueing system, with events and notation as described in Example 1. For this system, there are no events cancelled in every valid event sequence. For example, the event sequence $IBCBBCBB$ results in

$$A(IB) = \{B, C\}, \qquad A(IBC) = \{B\},$$

$$A(IBCB) = A(IBCBB) = A(IBCBBC) = A(IBCBBCB) = A(IBCBBCBB) = \{B, C\}.$$

Therefore, this particular event sequence does not yield a solution to INTERRUPT.

Suppose that the single server queueing system is embellished with the added feature that the server breaks down due to an overload (a new breakdown event, D), resulting in the cancellation of the service completion event and the scheduling of a repair event (a new repair event, R). For this model specification, the event sequence IBB results in $A(IBB) = \{B, C, D\}$ and $A(IBBD) = \{B, R\}$, hence $C \in A(IBB)\backslash\{D\}$, yet $C \notin A(IBBD)$. Therefore, for this model specification, this event sequence is a solution to INTERRUPT for $K \geq 2$. □

3. CONSEQUENCES AND IMPLICATIONS

The search problems STRONG PERMUTATION, WEAK PERMUTATION and INTERRUPT have been shown to be NP-hard. This implies that it is highly unlikely to obtain event sequences to solve these problems using a polynomial-time algorithm, unless $P = NP$. These results have a number of consequences for practitioners interested in building and analyzing simulation models.

Glasserman and Yao (1992a) define the concept of *monotonicity*. Informally, they state that this condition means that "the occurrence of

more events in the short run never leads to the execution of fewer events in the long run." Using the GSMP framework, they prove that verifying monotonicity is equivalent to verifying two conditions: permutability and non-interruptive. If there exists a valid event sequence that solves WEAK PERMUTATION, then the permutability condition does not hold. Similarly, if there exists a valid event sequence that solves INTERRUPT, then the non-interruptive condition is violated. They further define two other equivalent conditions within the framework of antimatroids (Glasserman and Yao, 1991). Since all of these problems are NP-hard search problems, then obtaining a polynomial-time algorithm to check the monotonicity condition in any given form can be done only if $P = NP$ (Garey and Johnson, 1979).

There are, however, some easy instances where these conditions can be readily verified. In a Simulation Graph, for example, *potential* interruptions can be identified as *cancelling edges* on the graph (Schruben and Yücesan, 1993), though one must still verify whether there exists a sequence of events that leads to an event cancellation. STRONG and WEAK PERMUTATION, however, are still hard to verify. Glasserman and Yao (1991), on the other hand, assert that STRONG and WEAK PERMUTATION automatically holds in a stochastic Petri net representation. In this case, however, INTERRUPT is hard to establish.

The complexity results in this paper have further practical implications. A special case of STRONG PERMUTATION is the *commuting condition*, which is a necessary condition for the applicability of the infinitesimal perturbation analysis technique in derivative estimation (Glasserman, 1991). Our results establish that the verification of the commuting condition is an intractable search problem. Note that this result was first obtained through the search problem ORDERING in Jacobson and Yücesan (1994).

The results for STRONG and WEAK PERMUTATION also imply that it is unlikely to construct a polynomial-time algorithm to determine the outcome of permutations in a sequence of events. It is therefore desirable to assign execution priorities to avoid arbitrary handling of simultaneously scheduled events in a simulation model implementation. Therefore, *a priori* determination of such priorities to ensure correct execution of model implementations is a difficult problem. This would be an important issue, for instance, in capacitated queueing systems, queueing systems with state-dependent routing as well as preemptive and non-preemptive priority and multiple customer-class queueing systems. Schruben (1983) and Sargent (1988) propose rules of thumb to signal potential problems with

simultaneously scheduled events. Som and Sargent (1989) develop conditions to identify when event execution priorities need to be established. Our results show that such conditions are impossible to verify.

Another implication of this result concerns the determination of when certain variance reduction techniques (VRT) will be successful. For instance, the effectiveness of common random numbers, antithetic variates, or control variates typically depends upon synchronization of events between pairs or sets of simulation runs. Such a synchronization requires that a permutation of a sequence of events have no impact on the resulting state. Our results then imply that it is unlikely to algorithmically determine whether satisfactory synchronization is achieved in such runs. As an easy instance, common random numbers together with inversion have been shown to minimize estimator variance provided that the monotonicity condition holds (Glasserman and Yao, 1992b). One should recall, however, that the verification of the commuting condition is a hard problem.

The practical implications of these results extend beyond the discrete event simulation modeling and analysis tasks and cover the design of actual discrete event dynamic systems (DEDS). For example, it is not possible to verify *a priori* the "robustness" of a manufacturing system in the non-interrupt sense. Increasing the rate of the arrival process (that is, the rate at which work is released into the system) does not necessarily translate into an increased system throughput, as higher workloads may unduly strain the system inducing machine breakdowns. Another such example is the investment into a quality improvement program. Current investment into quality assurance may not subsequently yield higher quality levels, as resources might be wasted or misallocated in the process. Solutions to INTERRUPT can provide a basis to identify such scenarios.

Similar problems arise with WEAK and STRONG PERMUTATION in actual DEDS. For example, alternate routing of jobs may not be possible due to the precedence constraints among the processing steps required by those jobs. This may be impossible even in a so-called flexible manufacturing system. A second example concerns the allocation of financial resources to investment opportunities where the profitability of the investment is contingent on the number and the sequence of subsequent events (e.g., changes in interest rates or introduction of new legislation). In general, solutions to WEAK PERMUTATION, STRONG PERMUTATION, and INTERRUPT can be used to gain insights into many types of DEDS. The complexity results for these three problems make finding these solutions particularly challenging.

4. CONCLUSIONS

This paper has introduced three new structural search problems for discrete event simulation models, and proven them to be NP-hard. The theoretical and practical implications of these results have been discussed.

These results bridge two areas of computer science and operations research, computational complexity and computer simulation, which enables several seemingly different problems in simulation modeling and analysis to be cast in a single unifying framework using the theory of computational complexity. In particular, such problems are equivalent or equally difficult, from the computational complexity point of view.

A consequence of the computational complexity results presented here is that algorithms that solve the three search problems are likely to be enumerative in nature. Such enumerative algorithms tend to execute in exponential time in the size of the problem instance. This, in turn, supports the development of polynomial-time heuristic procedures as well as the identification of special cases that are polynomially solvable. Further research is in progress to gain new insights from the three problems as well as to identify other related problems that may have an impact on the way discrete-event simulation models are constructed and analyzed.

REFERENCES

M. R. GAREY and D. S. JOHNSON, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman and Company, San Francisco, California, 1979.

P. GLASSERMAN, *Gradient Estimation via Perturbation Analysis,* Kluwer Academic Publishers Group. Dordrecht, The Netherlands, 1991.

P. GLASSERMAN and D. D. YAO, Algebraic Structure of Some Stochastic Discrete Event Systems, with Applications, *Discrete Event Dynamic Systems: Theory and Applications*, Vol. *1*(1), 1991, pp. 7-35.

P. GLASSERMAN and D. D. YAO, Monotonicity in Generalized Semi- Markov Processes, *Mathematics of Operations Research*, Vol. *17*(1), 1992a, pp. 1-21.

P. GLASSERMAN and D. D. YAO, Some Guidelines and Guarantees for Common Random Numbers, *Management Science*, Vol. *38*(6), 1992b, pp. 884-908.

S. H. JACOBSON and E. YÜCESAN, *On the Complexity of Verifying Structural Properties of Discrete Event Simulation Models*, Working Paper. INSEAD. Fontainebleau, France, 1994.

R. G. SARGENT, Event Graph Modeling for Simulation with an Application to Flexible Manufacturing Systems, *Management Science*, Vol. *34*(10), 1988, pp. 1231-1251.

L. SCHRUBEN, Simulation Modeling with Event Graphs, *Communications of the ACM*, Vol. *26*(11), 1983, pp. 957-963.

L. SCHRUBEN, *SIGMA: A Graphical Simulation System*, 2nd Edition. The Scientific Press. San Fransisco, CA, 1992.

L. SCHRUBEN and E. YÜCESAN, Modeling Paradigms for Discrete Event Simulation, *Operations Research Letters*, Vol. *13*, 1993, pp. 265-275.

T. K. SOM and R. G. SARGENT, A Formal Development of Event Graphsas an Aid to Structured and Efficient Simulation Programs, *ORSA Journal on Computing*, *1*(2), 1989, pp. 107-125.

E. YÜCESAN and S. H. JACOBSON, Building Correct Simulation Models is Difficult, *Proceedings of the 1992 Winter Simulation Conference*, 1992, 783-790.