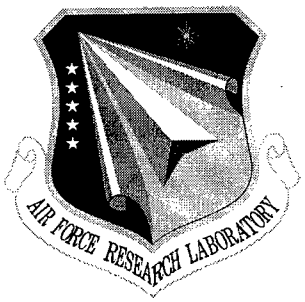
AFRL-IF-RS-TR-1999-245
Final Technical Report
November 1999



REAL-TIME SIMULATION TECHNOLOGIES FOR COMPLEX SYSTEMS

Boston University

Christos G. Cassandras and Wei-Bo Gong

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-1999-245 has been reviewed and is approved for publication.

APPROVED:

ALEX F. SISTI
Project Engineer

FOR THE DIRECTOR:

JAMES W. CUSACK, Chief
Information Systems Division
Information Directorate

If your address has changed or if you wish to be removed from the Air Force Research Laboratory Rome Research Site mailing list, or if the addressee is no longer employed by your organization, please notify AFRL/IFSB, 525 Brooks Road, Rome, NY 13441-4505. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	November 1999	Final May 97 - Apr 99

4. TITLE AND SUBTITLE

REAL-TIME SIMULATION TECHNOLOGIES FOR COMPLEX SYSTEMS

5. FUNDING NUMBERS

C - F30602-97-C-0125
PE - 62702F
PR - 459S
TA - 15
WU - N8

6. AUTHOR(S)

Christos G. Cassandras and Wei-Bo Gong

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Boston University
Dept of Manufacturing Engineering
Boston MA 02215

University of Massachusetts
Dept of Elec & Comp Engineering
Amherst MA 01003

8. PERFORMING ORGANIZATION REPORT NUMBER

N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

AFRL/IFSB
525 Brooks Road
Rome NY 13441-4505

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

AFRL-IF-RS-TR-1999-245

11. SUPPLEMENTARY NOTES

AFRL Project Engineer: Alex F. Sisti/IFSB/(315) 330-3983

12a. DISTRIBUTION AVAILABILITY STATEMENT

Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This report summarizes the work performed for "Real - Time Simulation Technologies for Complex Systems". The objective of the effort was to develop and study three novel complimentary directions that may be summarized as follows:

a. Speed up the inherently slow simulation process of complex systems by exploiting new concurrent simulation techniques.
b. Exploit the hierarchical nature of multi-resolution simulation models by decomposing them in ways which preserve statistical fidelity.
c. Explore new ways to extract metamodels for complex systems simulations, using neural networks.

14. SUBJECT TERMS

Concurrent simulation, Statistical Fidelity, Adaptive Resonance Theory Metamodeling, "Cascade Correlation Neural Networks (CCNN)"

15. NUMBER OF PAGES

132

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 INTRODUCTION

This report summarizes the work we have performed for the project entitled "Real-Time Simulation Technologies for Complex Systems." The objective of this effort has been to develop and study three novel complementary directions that may be summarized as follows:

1. Speed up the inherently slow simulation process of complex systems by exploiting new *concurrent* and *parallel* techniques.

2. Exploit the *hierarchical* structure in multi-resolution simulation models by decomposing them in ways which preserve *statistical fidelity.*

3. Explore new ways to extract *metamodels* for complex systems from simulation. We have been specifically targeting a radically different metamodeling approach using *neural networks.*

The scope of the project has been to develop specific methodologies and algorithms and test them on benchmark problems in C^4I application areas. Thus, appropriate simulation models were built, and algorithms based on the proposed new techniques were developed and tested. In many cases, the benchmark problems studied are the same or extensions of the ones developed during our previous project entitled "Enabling Technologies for Real-Time Simulation" (see also [5]).

As in our last report [5], we begin by briefly outlining some of the major challenges faced by modeling and simulation techniques for complex systems and the approaches we are following to address these challenges (Section 1.1). We then describe the organization of this report (Section 1.2).

1.1 Issues in Modeling and Simulation of Complex Systems

Simulation is widely recognized as one of the most versatile and general-purpose tools available today for modeling complex processes and solving problems in design, performance evaluation, decision making, and planning. This includes C^4I environments, where most problems confronted by designers and decision makers are of such complexity that their analysis and solution far surpass the scope of available analytical and numerical methods; this leaves simulation as the only alternative of "universal" applicability.

The importance of discrete event simulation has given rise to a number of commercially available software packages (e.g., SIMAN, SLAM, SIMULA, SIMSCRIPT, MODSIM, EXTEND) whose applicability ranges from very generic to highly specialized. However, the use of typical simulation software is limited by factors such as the following: (a) One must have thorough knowledge of the specific tool at a detailed technical level before attempting to use it in a modeling effort. (b) One must be an experienced programmer, in addition to a decision maker. (c) In order to make decisions based on simulation, one usually needs to run a large number of simulations and then carefully manage all output data collected on a case-by-case basis. (d) The field of simulation was developed primarily as a special branch of statistics involving dynamical phenomena. Manual handling and analysis of input/output data is still the norm, while design of interfaces, componentware interoperability, intelligent and automated analysis of output have been neglected. For example, the practice of Object Oriented Programming (OOP), with few exceptions, is still nascent in simulation languages despite the fact that the OOP idea actually originated in simulation. In addition, hardware advances, such as massively parallel computers and workstation networking, are only beginning to be noticed in simulation theory and practice. (e) The ultimate purpose of simulation is often system performance evaluation and optimization. However, simulation is notoriously computer time-consuming when it comes to parametric studies of system perfor-

mance. Unless substantial speedup of the performance evaluation process can be achieved, systematic performance studies of most real-world problems are beyond reach, even with supercomputers.

With this brief discussion in mind, we identify below some of the issues that we believe constitute the major challenges faced by simulation technology today, and introduce some key ideas which have been the subject of further study in this project.

1. **"What if" capability and concurrent simulation.** A major goal of any simulator is to provide the capability to explore a multitude of "what if" scenaria. The obvious way to obtain answers to N "what if" questions is to perform $(N+1)$ separate simulations: one for some baseline scenario and N additional ones for each "what if" question. If a typical simulation run takes T time units, this procedure requires a total of $(N+1)T$ time units. In *concurrent simulation*, this objective is met by performing a single baseline simulation, but endowing it with the capability to generate all $(N+1)$ desired simulations concurrently. This is accomplished by exploiting an intelligent sharing of data which results in a total simulation time of $(T+c) << (N+1)T$, where c represents the overhead corresponding to this "intelligent data sharing". Concurrent simulation can be carried out on any sequential computer. The universal applicability of this approach has been the subject of our work during the previous and current projects. Although this issue has been resolved through the development of the *Time Warping Algorithm* (TWA) (see [7]), one of the findings of this project is that its computational efficiency significantly depends on how it is implememented on any given computer platform.

2. **Hierarchical simulation and statistical fidelity.** In modeling complex systems it is impossible to mimic every detail through detailed simulation. The common approach is to divide the whole system hierarchically into modules with different simulation

resolutions. The lower, high resolution, level simulator generates reports which are then taken as inputs for the higher level simulator. Current practice is to use the mean values of variables from the lower level reports as the input to the higher level. This implies that significant statistical information (i.e., *statistical fidelity*) is lost in this process, resulting in potentially completely inaccurate results. Especially when the ultimate output of the simulation process is of the form 0 or 1 (e.g., "lose" or "win" a combat), such errors can provide the exact opposite of the real output. Our effort has been directed at developing an interface between the two simulation levels to preserve the statistics to the maximum extent that the available computing power allows. In our previous project [5] we initiated a study of an approach based on *clustering* or *path bundle grouping* which was further pursued in this project.

3. **Metamodeling through Neural Networks.** The main idea of metamodeling is to extract as much information from simulation as possible and process it so as to build a *surrogate model* of the system of interest which is much simpler (yet accurate) to work with. This is essentially analogous to constructing a function $F(x_1, \ldots, x_N)$ from only selected values observed under selected combinations of values of $x_1, \ldots, x_N$. The problem, of course, is that the actual function we are trying to approximate with $F(x_1, \ldots, x_N)$ is unknown. The most common approach is to try and build a *polynomial* expression. This is often inadequate because if the shape of the actual curve corresponding to $F(x_1, \ldots, x_N)$ includes sudden jumps and asymptotic behavior (which is very often the case from experience), then polynomial fits to such curves are known to be poor. Thus, obtaining a metamodeling device of "universal" applicability, i.e., one capable of generating functions of virtually arbitrary complexity, remains an open issue. This project has explored *neural networks* as offering this capability, including some benchmark models and problems that have been analyzed with these techniques.

1.2 Report Organization

The contents of this report may be outlined as follows.

- **Section 2:** The basic theoretical framework for explaining the concurrent simulation algorithms we have developed is first reviewed. Based on this framework, a general concurrent simulation approach was developed under our previous project, where a detailed *Time Warping Algorithm* (TWA) was also introduced. The concept of "speedup" was used in order to provide a clear quantitative measure of the improvement provided by concurrent simulation over conventional simulation techniques. This report includes the following new findings from our project: (a) extensions of the TWA that enhance its range of applications, (b) further investigations based on the speedup factor and some explicit numerical results, and (c) a study of the statistical significance of estimates obtained through the TWA. In addition, Section 2.7 describes the use of Concurrent Simulation in complex resource allocation problems requiring answers to a large number of 'what if" questions in a near-real-time setting to support decision making in a C^4I setting.

- **Section 3** The stochastic fidelity preservation issue in multi-resolution models of complex systems is discussed in detail. The *Path Bundle Grouping* (PBG) approach is described as a way to maintain stochastic fidelity in hierarchical simulations. The PBG approach is related to the theory of cluster analysis. We have reviewed different aspects of this theory in order to compare them to the Adaptive Resonance Theory (ART) neural network we have developed for the purpose of automating the task of path bundle grouping. We also report on an interesting "real-world" application we encountered in the course of this project.

- **Section 4** We review the main concepts involved in using neural networks as universal function approximators. A particular type of neural network, the *Cascade Correlation*

Neural Network (CCNN) was developed for the purposes of this project for its ability to build itself to an appropriate size as part of its learning process during the training phase. This was used in [5] on a testbed model using the *Tactical Electronic Reconnaissance Simulator*(TERSM). This section includes several numerical results and comparisons with polynomial metamodels previously derived, as a continuation of a study initiated earlier and presented in [5]. A new benchmark problem is studied based on an *Aircraft Refueling and Maintenance System* (ARMS) model.

- **Section 5** We present the main conclusions of our study, including lessons learned and recommendations. We also outline our ongoing work and some future research directions.

2 CONCURRENT SIMULATION

A major objective of this project is motivated by the time-consuming nature of system performance exploration through simulation: to obtain answers to N "what if" questions, $(N+1)$ simulations are needed. Therefore, our goal is the following: *From a single simulation, obtain answers to all N "what if" questions simultaneously.* The main idea behind the approach we used to solve this problem is to observe the evolution of a single sample path of the Discrete Event System (DES) under study, called the *nominal* sample path, as it operates under some parameter. As the sample path evolves, observed data (e.g., event occurrences and their corresponding occurrence times) are processed to concurrently construct the set of sample paths that would have resulted if the system had operated under a set of different (hypothetical) parameters. Using these "concurrently constructed" hypothetical sample paths, it is possible to "concurrently estimate" the corresponding performance measures which can be used in the design of the actual system.

In this section we review the principles of concurrent simulation for solving the fundamen-

tal "sample path constructability problem" (Section 2.1). We use the concept of a stochastic timed state automaton as a modeling framework for general DES, allowing us to describe a procedure for constructing sample paths of DES, a formal way of characterizing the function of any discrete event simulation software package (Section 2.2). In Section 2.3 we summarize the concurrent simulation method developed in our previous project [5] for solving the sample path constructability problem, culminating with the *Time Warping Algorithm* (TWA). In Section 2.4 we summarize extensions that we were able to obtain during the course of this project. A significant part of our effort was also directed at investigating the computational efficiency of the TWA and our findings are summarized in Section 2.4. In Section 2.5, we also address the issue of statistical significance of estimates obtained through the TWA. In an Appendix to this Final Report we have included a copy of a recently published paper entitled "Concurrent Sample Path Analysis of Discrete Event Systems", which presents the "concurrent simulation" algorithm and its analysis in more detail.

2.1 The Sample Path Constructability Problem

We adopt the modeling framework of a *stochastic timed state automaton* $(\mathcal{E},\mathcal{X},\Gamma,f,x_0)$ [4] to characterize the function of any discrete event simulator. Here, $\mathcal{E}$ is a countable event set, $\mathcal{X}$ is a countable state space, and $\Gamma(x)$ is a set of feasible (or enabled) events, defined for all $x \in \mathcal{X}$ such that $\Gamma(x) \subseteq \mathcal{E}$. The state transition function $f(x,e)$ is defined for all $x \in \mathcal{X}$, $e \in \Gamma(x)$, and specifies the next state resulting when e occurs at state x. Finally, x_0 is a given initial state. The definition is easily modified to $(\mathcal{E},\mathcal{X},\Gamma,p,p_0)$ in order to include probabilistic state transition mechanisms. In this case, the state transition probability $p(x';x,e')$ is defined for all $x,x' \in \mathcal{X}$, $e' \in \mathcal{E}$, and is such that $p(x';x,e') = 0$ for all $e' \notin \Gamma(x)$. In addition, $p_0(x)$ is the pmf $P[x_0 = x]$, $x \in \mathcal{X}$, of the initial state x_0.

Assuming the cardinality of the event set $\mathcal{E}$ is N, the input to the system is a set of

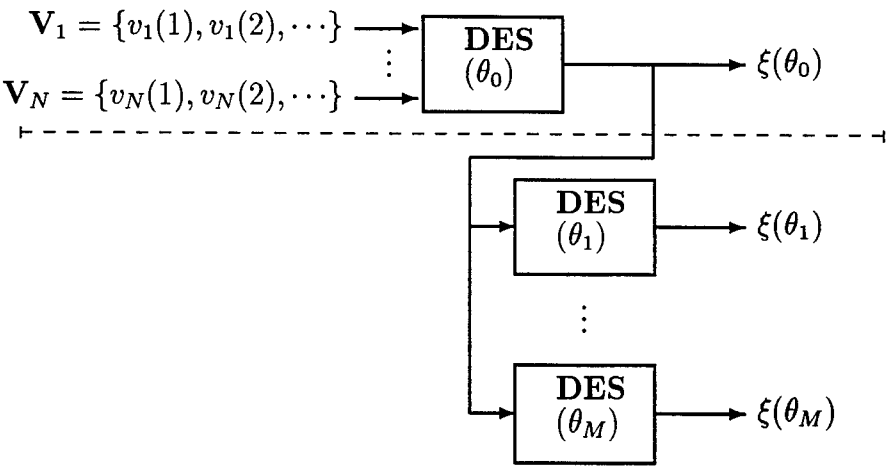


Figure 1: The sample path constructability problem for DES

event lifetime sequences $\{\mathbf{V}_1, \cdots, \mathbf{V}_N\}$, one for each event, where $\mathbf{V}_i = \{v_i(1), v_i(2), \cdots\}$ is characterized by some arbitrary distribution. Under some system parameter θ_0, the output is a sequence $\xi(\theta_0) = \{(e_k, t_k),\ k = 1, 2, \cdots\}$ where $e_k \in \mathcal{E}$ is the kth event and t_k is its corresponding occurrence time (see Figure 1). Based on any observed $\xi(\theta_0)$, we can evaluate $L[\xi(\theta_0)]$, a sample performance metric for the system. For a large family of performance metrics of the form $J(\theta_0) = E[L[\xi(\theta_0)]]$, $L[\xi(\theta_0)]$ is therefore an estimate of $J(\theta_0)$. Defining a set of parameter values of interest $\{\theta_0, \theta_1, \cdots, \theta_M\}$, the sample path constructability problem is:

> *For a DES under θ_0, construct all sample paths $\xi(\theta_1), \cdots, \xi(\theta_M)$ given a realization of lifetime sequences $\mathbf{V}_1, \cdots, \mathbf{V}_N$ and the sample path $\xi(\theta_0)$.*

For simplicity, we assume that the system we are modeling satisfies the following three assumptions. Extensions allowing the relaxation of these assumptions are possible and are described in [7] (see Appendix).

- *(A1) Feasibility Assumption*: Let x_n be the state of the DES after the occurrence of

the nth event. Then, for any n, there exists at least one $r > n$ such that $e \in \Gamma(x_r)$ for any $e \in \mathcal{E}$.

- *(A2) Invariability Assumption*: Let $\mathcal{E}$ be the event set under the nominal parameter θ_0 and let $\mathcal{E}_m$ be the event set under $\theta_m \neq \theta_0$. Then, $\mathcal{E}_m = \mathcal{E}$.

- *(A3) Similarity Assumption*: Let $G_i(\theta_0), i \in \mathcal{E}$ be the event lifetime distribution for the event i under θ_0 and let $G_i(\theta_m), i \in \mathcal{E}$ be the corresponding event lifetime distribution under θ_m. Then, $G_i(\theta_0) = G_i(\theta_m)$ for all $i \in \mathcal{E}$.

Assumption *A1* guarantees that in the evolution of any sample path all events in $\mathcal{E}$ will always become feasible at some point in the future. Note that a DES with an irreducible state space immediately satisfies this condition. Assumption *A2* states that changing a parameter from θ_0 to some $\theta_m \neq \theta_0$ does not alter the event set $\mathcal{E}$. More importantly, *A2* guarantees that changing to θ_m does not introduce any new events so that all event lifetimes for all events can be observed from the nominal sample path. Finally, assumption *A3* guarantees that changing a parameter from θ_0 to some $\theta_m \neq \theta_0$ does not affect the distribution of one or more event lifetime sequences. This allows us to use exactly the same lifetimes that we observe in the nominal sample path to construct the perturbed sample path. In other words, our analysis focuses on *structural* system parameters rather that *distributional* parameters. However, it is possible to handle the latter at the expense of some computational cost, as described in Section 2.4.

2.2 Sample Path Construction Procedure

First, let $\xi(n, \theta) = \{e_j : j = 1, \cdots, n\}$, with $e_j \in \mathcal{E}$, be the sequence of events that constitute the observed sample path up to n total events. Although $\xi(n, \theta)$ is clearly a function of the parameter θ, we will write $\xi(n)$ to refer to the observed sample path. Next we define the

score of an event $i \in \mathcal{E}$ in a sequence $\xi(n)$, denoted by $s_i^n = [\xi(n)]_i$, to be the non-negative integer that counts the number of instances of event i in this sequence.

We introduce two additional variables, t_n to be the time when the nth event occurs, and $y_i(n)$, $i \in \Gamma(x_n)$, to be the residual lifetime of event i after the occurrence of the nth event (i.e., it is the time left until event i occurs). On a particular sample path, just after the nth event occurs the following information is known: the state x_n from which we can determine $\Gamma(x_n)$, the time t_n, the residual lifetimes $y_i(n)$ for all $i \in \Gamma(x_n)$, and all event scores s_i^n, $i \in \mathcal{E}$. The following equations describe the dynamics of the timed state automaton.

step 1: Determine the smallest residual lifetime among all feasible events at state x_n, denoted by y_n^*:

$$y_n^* = \min_{i \in \Gamma(x_n)} \{y_i(n)\} \tag{1}$$

step 2: Determine the triggering event:

$$e_{n+1} = arg \min_{i \in \Gamma(x_n)} \{y_i(n)\} \tag{2}$$

step 3: Determine the next state:

$$x_{n+1} = f(x_n, e_{n+1}) \tag{3}$$

step 4: Determine the next event time:

$$t_{n+1} = t_n + y_n^* \tag{4}$$

step 5: Determine the new residual lifetimes for all new feasible events $i \in \Gamma(x_{n+1})$:

$$y_i(n+1) = \begin{cases} y_i(n) - y_n^* & \text{if } i \neq e_{n+1} \text{ and } i \in \Gamma(x_n) \\ v_i(s_i^n + 1) & \text{if } i = e_{n+1} \text{ or } i \notin \Gamma(x_n) \end{cases} \quad \text{for all } i \in \Gamma(x_{n+1}) \tag{5}$$

step 6: Update the event scores:

$$s_i^{n+1} = \begin{cases} s_i^n + 1 & \text{if } i = e_{n+1} \\ s_i^n & \text{otherwise} \end{cases} \tag{6}$$

Equations (1)-(6) describe the sample path evolution of a timed state automaton. These equations apply to both the observed and any constructed sample paths through concurrent simulation.

2.3 The Time Warping Algorithm (TWA)

The Time Warping Algorithm is a specific procedure for accomplishing concurrent simulation. It was introduced and described in our previous project [5], so that we limit ourselves here to a brief review.

We begin by summarizing the necessary notation. We use $\hat{\xi}(k) = \{\hat{e}_j : j = 1, \cdots, k\}$ to denote any constructed sample path under a different value of the parameter θ, where k is the number of events in that path. It is important to realize that k is actually a function of n, the number of event observed on the nominal path. This is because the constructed sample path is coupled to the observed sample path through the observed event lifetimes; however, for the sake of notational simplicity, we refrain from continuously indicating this dependence. The score of event i in a constructed sample path is denoted by $\hat{s}_i^k = [\hat{\xi}(k)]_i$. In what follows, all quantities with the symbol " $\hat{\cdot}$ " refer to a typical constructed sample path.

Associated with every event type $i \in \mathcal{E}$ in $\xi(n)$ is a sequence of s_i^n event lifetimes

$$\mathbf{V}_i(n) = \{v_i(1), \cdots, v_i(s_i^n)\} \quad \text{for all } i \in \mathcal{E}$$

The corresponding set of sequences in the constructed sample path is:

$$\hat{\mathbf{V}}_i(k) = \{v_i(1), \cdots, v_i(\hat{s}_i^k)\} \quad \text{for all } i \in \mathcal{E}$$

which is a subsequence of $\mathbf{V}_i(n)$ with $k \leq n$. In addition, we define the following sequence of lifetimes:

$$\tilde{\mathbf{V}}_i(n,k) = \{v_i(\hat{s}_i^k + 1), \cdots, v_i(s_i^n)\} \quad \text{for all } i \in \mathcal{E}$$

which consists of all event lifetimes that are in $\mathbf{V}_i(n)$ but not in $\hat{\mathbf{V}}_i(k)$.

Next, define the set

$$A(n,k) = \left\{i : i \in \mathcal{E}, s_i^n > \hat{s}_i^k\right\} \tag{7}$$

which is associated with $\tilde{\mathbf{V}}_i(n,k)$ and consists of all events i whose corresponding sequence $\tilde{\mathbf{V}}_i(n,k)$ contains at least one element. Thus, every $i \in A(n,k)$ is an event that has been observed in $\xi(n)$ and has at least one lifetime that has yet to be used in the coupled sample path $\hat{\xi}(k)$. Hence, $A(n,k)$ should be thought of as the set of *available* events to be used in the construction of the coupled path.

Finally, we define the following set, which is crucial in our approach:

$$M(n,k) = \Gamma(\hat{x}_k) - (\Gamma(\hat{x}_{k-1}) - \{\hat{e}_k\}) \tag{8}$$

where, clearly, $M(n,k) \subseteq \mathcal{E}$. Note that $\hat{e}_k$ is the triggering event at the $(k-1)$th state visited in the constructed sample path. Thus, $M(n,k)$ contains all the events that are in the feasible event set $\Gamma(\hat{x}_k)$ but not in $\Gamma(\hat{x}_{k-1})$; in addition, $\hat{e}_k$ also belongs to $M(n,k)$ if it happens that $\hat{e}_k \in \Gamma(\hat{x}_k)$. Intuitively, $M(n,k)$ consists of all *missing* events from the perspective of the constructed sample path when it enters a new state $\hat{x}_k$: those events already in $\Gamma(\hat{x}_{k-1})$ which were not the triggering event remain available to be used in the sample path construction as long as they are still feasible; all other events in the set are "missing" as far as residual lifetime information is concerned.

With this notation, we now present the *Time Warping Algorithm (TWA).*

Time Warping Algorithm (TWA):

1. INITIALIZE

$n := 0$, $k := 0$, $t_n := 0$, $\hat{t}_k := 0$, $x_n := x_0$, $\hat{x}_k = \hat{x}_0$,

$y_i(n) = v_i(1)$ for all $i \in \Gamma(x_n)$, $s_i^n = 0$,$\hat{s}_i^k = 0$ for all $i \in \mathcal{E}$,

$M(0,0) := \Gamma(\hat{x}_0)$, $A(0,0) := \emptyset$

2. WHEN EVENT e_n IS OBSERVED:

2.1 Use (1)-(6) to determine e_{n+1}, x_{n+1}, t_{n+1}, $y_i(n+1)$ for all $i \in \Gamma(x_{n+1})$, s_i^{n+1} for all $i \in \mathcal{E}$.

2.2 Add the e_{n+1} event lifetime to $\tilde{\mathbf{V}}_i(n+1,k)$:

$$\tilde{\mathbf{V}}_i(n+1,k) = \begin{cases} \tilde{\mathbf{V}}_i(n,k) + v_i(s_i^n + 1) & \text{if } i = e_{n+1} \\ \tilde{\mathbf{V}}_i(n,k) & \text{otherwise} \end{cases}$$

2.3 Update the available event set $A(n,k)$: $A(n+1,k) = A(n,k) \cup \{e_{n+1}\}$

2.4 Update the missing event set $M(n,k)$: $M(n+1,k) = M(n,k)$

2.5 IF $M(n+1,k) \subseteq A(n+1,k)$ then Goto **3.** ELSE set $n \leftarrow n+1$ and Goto **2.1.**

3. TIME WARPING OPERATION:

3.1 Obtain all missing event lifetimes to resume sample path construction at state $\hat{x}_k$:

$$\hat{y}_i(k) = \begin{cases} v_i(\hat{s}_i^k + 1) & \text{for } i \in M(n+1,k) \\ \hat{y}_i(k-1) & \text{otherwise} \end{cases}$$

3.2 Use (1)-(6) to determine $\hat{e}_{k+1}$, $\hat{x}_{k+1}$, $\hat{t}_{k+1}$, $\hat{y}_i(k+1)$ for all $i \in \Gamma(\hat{x}_{k+1}) \cap (\Gamma(\hat{x}_k) - \{\hat{e}_{k+1}\})$, $\hat{s}_i^{k+1}$ for all $i \in \mathcal{E}$.

3.3 Discard all used event lifetimes:

$$\tilde{\mathbf{V}}_i(n+1,k+1) = \tilde{\mathbf{V}}_i(n+1,k) - v_i(\hat{s}_i^k + 1) \text{ for all } i \in M(n+1,k)$$

3.4 Update the available event set $A(n+1,k)$:

$$A(n+1,k+1) = A(n+1,k) - \left\{i : i \in M(n+1,k),\ \hat{s}_i^{k+1} = s_i^{n+1}\right\}$$

3.5 Update the missing event set $M(n+1,k)$:

$$M(n+1,k+1) = \Gamma(\hat{x}_{k+1}) - (\Gamma(\hat{x}_k) - \{\hat{e}_{k+1}\})$$

3.6 IF $M(n+1,k+1) \subseteq A(n+1,k+1)$ then $k \leftarrow k+1$ and Goto **3.1.** ELSE $k \leftarrow k+1$, $n \leftarrow n+1$ and Goto **2.1**.

The computational requirements of TWA are minimal (adding and subtracting elements to sequences, simple arithmetic, and checking the subset condition in steps 2.5 and 3.6 above. Rather, it is the storage of additional information that constitutes the major cost of the algorithm.

2.4 Extensions of the TWA

In section 2.1 we stated three assumptions that were made to simplify the development of our approach and keep the TWA notationally simple. It turns out that we can extend the application of TWA by relaxing these assumptions at the expense of some extra work.

In *A2* we assumed that changing a parameter from θ_0 to some $\theta_m \neq \theta_0$ does not alter the event set $\mathcal{E}$. Clearly, if the new event set, $\mathcal{E}_m$ is such that $\mathcal{E}_m \subseteq \mathcal{E}$, the development and analysis of *TWA* is not affected. If, on the other hand, $\mathcal{E} \subset \mathcal{E}_m$, this implies that events required to cause state transitions under θ_m are unavailable in the observed sample path, which make the application of our algorithm impossible. In this case, one can introduce *phantom* event sources which generate all the unavailable events, provided that the lifetime distributions of these events are known. The idea of phantom sources can also be applied

to DES that do not satisfy *A1*. In this case, if a sample path remains suspended for a long period of time, then a phantom source can provide the required event(s) so that the sample path construction can resume.

In *A3* we assumed that changing a parameter from θ_0 to some $\theta_m \neq \theta_0$ does not affect the distribution of one or more event lifetime sequences. This assumption is used in step 2.2 of the TWA where the observed lifetime $v_i(s_i^n + 1)$ is directly suffix-added to the sequence $\tilde{\mathbf{V}}_i(n+1, k)$. Note that this problem can be overcome by transforming observed lifetimes $\mathbf{V}_i = \{v_i(1), v_i(2), \cdots\}$ with an underlying distribution $G_i(\theta_0)$ into samples of a similar sequence corresponding to the new distribution $G_i(\theta_m)$ and then suffix-add them in $\tilde{\mathbf{V}}_i(n+1, k)$. This is indeed possible, if $G_i(\theta_0)$, $G_i(\theta_m)$ are known, at the expense of some additional computational cost for this transformation (for example, see [4]). One interesting special case arises when the parameter of interest is a scale parameter of some event lifetime distribution (e.g., it is the mean of a distribution in the Erlang family). Then, simple rescaling suffices to transform an observed lifetime v_i under θ_0 into a new lifetime $\hat{v}_i$ under θ_m:

$$\hat{v}_i = (\theta_m/\theta_0)v_i$$

Finally note that in a simulation environment it is possible to eliminate the overhead which is due to checking the subset condition in step **2.5**. In order to achieve this we need to eliminate the coupling between the observed and constructed sample paths. Towards this goal, we can simulate the nominal sample but rather than disposing the event lifetimes we save them all in memory. Once the simulation is done, we simulate one by one all the perturbed sample paths exactly as we do with a "brute force" simulation scheme but rather than generating the required random variates we read them directly from the computer memory. In this way we trade off computer memory for higher speedup. A quantification of this tradeoff is the subject of ongoing research.

2.5 Comparative Speedup Analysis

To define the *speedup factor* associated with concurrent simulation, suppose that the sample path constructed through our coupling approach were instead generated by a separate simulation whose length is defined by N total events. Let T_N be the time it takes (in CPU time units) to complete such a simulation run. Further, suppose that when the nominal simulation is executed with TWA as part of it, the total time is given by $T_N^o + \tau_K$, where T_N^o is the simulation time *without* the *TWA* and τ_K is the additional time involved in the concurrent construction of a sample path with $K \leq N$ events. We then define the *speedup factor* due to TWA as

$$S = \frac{T_N/N}{\tau_K/K} \tag{9}$$

Thus, if a separate simulation (in addition to the one for the observed sample path) were to be used to generate a sample path under a new value of the parameter of interest, the computation time per event is T_N/N. If, instead, we use the TWA in conjunction with the observed path, no such separate simulation is necessary, but the additional time per event imposed by the approach is τ_K/K, where $K \leq N$ in general.

A number of simulation results that include speedup computations relative to "brute force" simulation (i.e., separately simulating N parameter settings) were included in [5]. Another interesting issue we have addressed is that of comparing the speedup performance of the TWA to two other known methods for concurrent simulation, Augmented System Analysis *ASA* and the Standard Clock method *SC*, both of which are limited to Markovian event processes. As expected, both techniques can achieve higher speedup than the TWA, as indicated in Figure 2 for an $M/M/1/K$ system studied over a range of values for K. *ASA* can achieve a speedup of up to 30, considerably higher than both *SC* and *TWA*, however, it is only applicable to systems with exponential event lifetime distributions (with the possibility of one non-Markovian event process, as noted in section 1). *SC* can achieve

a speedup of up to 8, however it is also limited to exponential event lifetime distributions (unless approximations are used for systems with general distributions). In Figure 2, the speedup for the *TWA* turns out to be in the vicinity of 2.

Finally, note that from the definition of speedup (9), one would expect that it is not a function of the number of concurrently constructed sample paths. Intuitively, suppose that one is interested in concurrently constructing M sample paths. Therefore, using brute force it would take MT_N time units to generate MN events, while using any other constructability technique it would require $M\tau_K$ time units to construct MK events. Hence, the speedup factor should be independent of M. However, the number of constructed events depends on the parameter settings as well. For this reason, the number of constructed events is given by $MK - K_0$ (not MK). Therefore,

$$S = \frac{MT_N/MN}{M\tau_K/(MK - K_0)} = \frac{T_N}{\tau_K}\left(\frac{K}{N} - \frac{K_0}{NM}\right)$$

which implies that it approaches a constant as M becomes larger.

2.6 Statistical Significance of Estimates obtained through TWA

As indicated earlier, the constructed sample path may remain suspended for extended periods of time while waiting for one or more of the missing events. This in turn, implies that while the length of the observed sample path (N) is long enough to guarantee that the observed measures are statistically significant, the length of the constructed sample path (K) may not become long enough to provide such accuracy. Figure 3 shows the ratio (K/N) for an $M/M/1/K$ queueing system when there are four classes of customers and the observed sample path has five buffer slots (i.e., $K = 5$). First note that when all events occur with similar frequency, the K/N ratio converges quickly ($M/M/1/7$ curve), whereas, when there is a rare event (class 4 arrival) often the constructed sample path is forced to wait for long

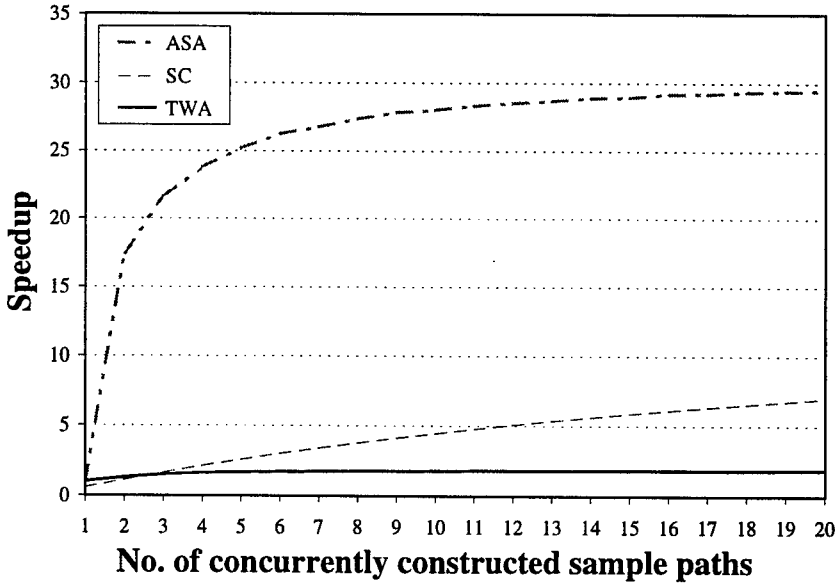


Figure 2: Speedup of *ASA*, *SC* and *TWA*, for an $M/M/1/K$ system.

intervals until the rare event is observed which causes the K/N ratio to become small. Once the missing event is observed, a large number of events may be immediately processed allowing K/N to increase again. This results in the initial large oscillations observed in Figure 3 which eventually diminish as N grows larger.

In addition, note that the parameter setting also affect the K/N ratio. In the case of a single buffer slot ($M/M/1/1$), the blocking probability is much larger than in the observed sample path, therefore, several observed departure events are not constructed because the corresponding customer was lost. For this reason, K/N converges to a value less than one (in this case 0.85). On the other hand, when the constructed sample path has nine slots, the observed and constructed sample paths have comparable blocking probabilities therefore most of the observed event are also constructed, so the K/N ratio is closer to one.

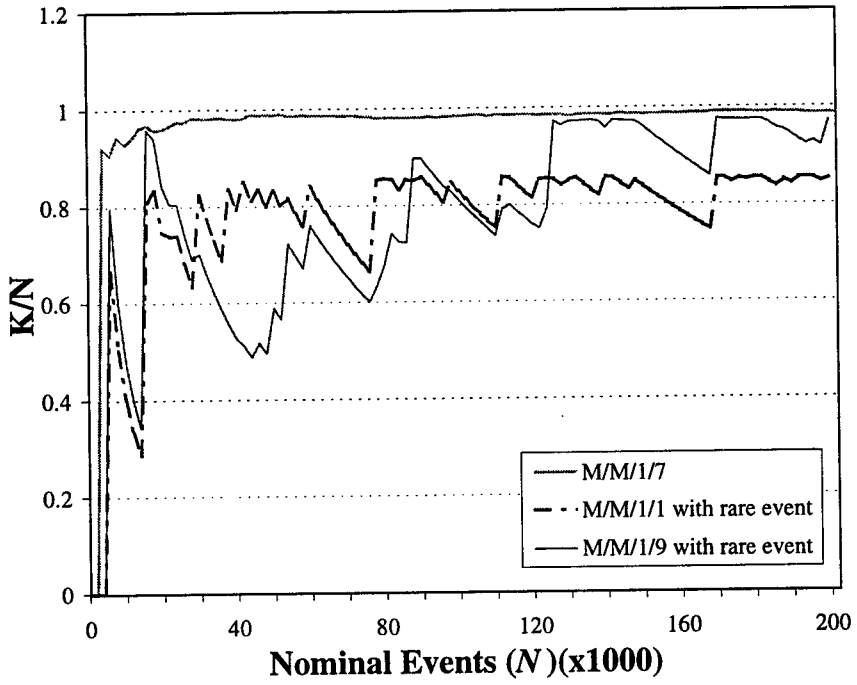


Figure 3: K/N ratio for system 1.

2.7 Resource Allocation and Concurrent Simulation

Resource allocation is a basic problem one encounters in numerous aspects of C^4I systems (e.g., allocating ammunition or platforms to different missions, sensors to different platforms, communication bandwidth to battle space entities). The mathematical representation of such resource allocation problems is as follows. Let $r \in Z_+^N$ be the decision vector or "state". In general, there is a set of feasible states denoted by A_d such that $r \in A_d$ represents a constraint. For example, in a typical resource allocation problem, r_i denotes the number of resources that user i is assigned subject to a capacity constraint of the form $A_d = \{r : \sum_{i=1}^N r_i = K\}$. In a stochastic setting, let $L_d(r, \omega)$ be the cost incurred over a specific sample path (denoted by ω) and $J_d(r)$ be the expected cost of a system operating under r. Then, the discrete

optimization problem we are interested in is the determination of $r^* \in A_d$ such that

$$J_d(r^*) = \min_{r \in A_d} J_d(r) = \min_{r \in A_d} E_\omega[L_d(r, \omega)] \tag{10}$$

In general, this is a notoriously hard stochastic integer programming problem. Even in a deterministic setting, where we may set $J_d(r) = L_d(r, \omega)$, this class of problems is NP-hard. When the system operates in a stochastic environment (e.g., in a resource allocation setting users request resources at random time instants or hold a particular resource for a random period of time) and no closed-form expression for $E_\omega[L_d(r, \omega)]$ is available, the problem is further complicated by the need to estimate $E_\omega[L_d(r, \omega)]$. This generally requires Monte Carlo simulation or direct measurements made on the actual system, approaches which are generally computationally intensive and slow for the purpose of rapid decision-making.

With this motivation, we have worked toward developing resource allocation algorithms that can take explicit advantage of Concurrent Simulation methods such as the TWA discussed in the previous section. We outline below our basic approach and include in the Appendix a paper [16] that provides technical details and sample numerical results. The key idea is to transform the original discrete set A_d into a continuous set over which a "surrogate" optimization problem is defined and subsequently solved. At every step of the continuous optimization process, the continuous state obtained is mapped back into a feasible discrete state; based on a realization under this feasible state, new sensitivity estimates are obtained that drive the surrogate problem to yield the next continuous state. The proposed scheme, therefore, involves an interplay of sensitivity-driven iterations and continuous-to-discrete state transformations. The key issue then is to show that when (and if) an optimal allocation is obtained in the continuous state space, the transformed discrete state is in fact r^* in (10).

Clearly, he ability to obtain sensitivity estimates with respect to discrete decision variables is a critical component of this approach. Concurrent Simulation methods are ideally

suited to meet this objective and provide the synergy required between real-time decision making approaches and the need for rapid simulation-based information to support these approaches. The Appendix provides additional material that details this approach as an application of Concurrent Simulation.

3 STOCHASTIC FIDELITY IN MULTI-RESOLUTION SIMULATION MODELS

In modeling complex systems through simulation, such as theater-level combat or large scale high speed communication networks, it is impossible to mimic every detail. The most common approach is to hierarhically decompose the whole system into modules with different simulation resolutions. The low resolution module consists of some coarse-grained equations, which are used to model the large scale, low resolution behavior. The coefficients of these "low resolution" equations are derived from the high resolution module, which executes some detailed, smaller scale simulations. The interfaces between these modules are critical. Common practice is to use the overall average of the high resolution results as the input to the low resolution simulator. This neglects the facts that (a) the high resolution results could have dramatically different *qualitative* features and should not be just "averaged" *quantitatively*, and (b) even when the high resolution simulation paths are qualitatively similar, the *variance* of the paths still needs to be accounted for in the low resolution module. To address (a) our approach is to cluster the high resolution combat simulation sample paths according to their features and use the averages over each cluster as inputs for the low resolution module. To address (b) our approach is to derive new low resolution dynamics that would account for the higher moments of the high resolution simulation paths. In the following, we first discuss the issue of clustering large dimensional data.

Clustering. The problem of systematically clustering data has drawn considerable research attention. Later in this section, we give a brief review of some classical clustering algorithms. These algorithms are too time-consuming when the data vector dimensions are high. In this project, we have considered the neural network (NN) technology, specifically a NN type known as ART NN, where ART stands for *Adaptive Resonance Theory*, as developed by Carpenter and Grossberg [3]. The key component in the ART NN design is a feedback mechanism that stabilizes the learning process. The input data is mixed with the trial feature for competitive learning, forming a feedback control loop where the input data is the external input, while the trial feature is the output which is fed back to mix with the input to stabilize the system. The key for success is of course the tuning of the feedback gain! This is motivated by human learning processes: enforcing already learned prototypes on the new data. It turns out that this is quite successful in ART clustering. Some numerical results are reported in [17] and included in the Appendix.

The ART NN is designed to deal with a classical dilemma in data clustering: if the requirement for distinct high resolution paths is too tight, then the resulting number of clusters will be too large to provide any benefit; if the requirement is too loose, then the resulting clusters will not in fact contain any characteristic features of the high resolution paths they encompass. The ART NN resolves this dilemma in this way: it tries to learn the similarities between high resolution paths, and create "prototype" clusters; as the process going on, it matches further data with existing prototypes. If no match is found, then a new prototype is created. Thus, previously learned information is never eroded by new knowledge.

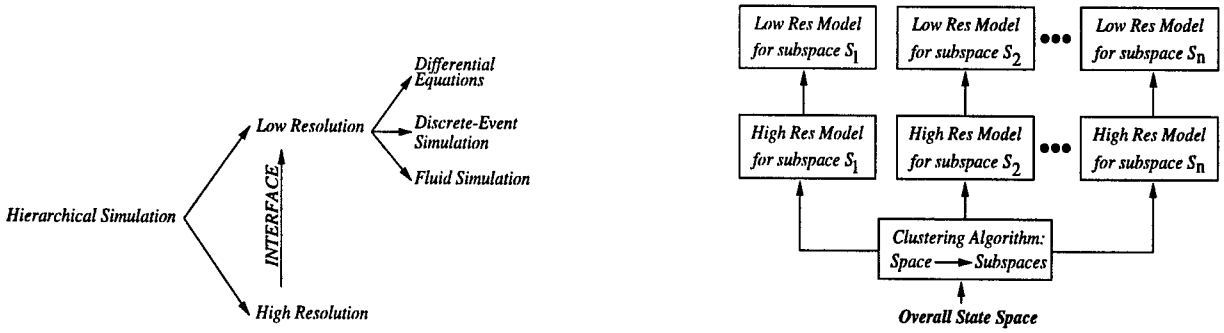
Deriving low resolution dynamics. We now discuss the issue of deriving low resolution dynamics, in which, the variability of the high resolution simulation paths is considered. Here is the basic idea: given a set of low resolution differential equations, these equations are based on the assumption that the high resolution output is deterministic. The high

resolution simulation is used to determine the coefficients of the low resolution differential equations. Since the actual high resolution paths are random, the low resolution dynamics should be modified. In the case of combat simulation, our proposed such modification is based on the analysis of the basic Lanchester equation. However, the principle is generally applicable for hierarchical simulation models.

The two ideas described above are of general importance. Combat modeling is just a typical example of a multi-time/space-scale system. Multiple-time-scale phenomena are encountered in numerous fields, including computer networks, manufacturing engineering, real-time systems, battlefield (combat) simulation, and physics. For example, in high-speed networks, traffic sources operate on at least three different time scales: connections for seconds to minutes, bursts within a connection for tens to hundreds of milliseconds, packets within a burst of the order of tens to hundreds of microseconds per packet. These present a tremendous challenge to the performance analyst. For instance, to study algorithms for establishing network connections in teleconferences (scale of minutes), one must capture the essential effects on those connections of bursty packets (scale of milliseconds). As another example, the lifetime of real-time processors can be several thousand hours; however, when a processor fails, fault detection, recovery, and the consequent switchover to another processor must take place in a few milliseconds. The life time of a processor has a quite different time-scale from the failure detection and recovery process. A third example, from control theory, is controlling the position of a rotating flexible steel shaft pinned at one end, such as a flexible robotic manipulator. The slow subsystem would correspond to the motion of the center of mass as the object rotates, while the fast subsystem would correspond to the vibrational motion of the flexible shaft or the position of the free end of the vibrating shaft. Simulating such systems using traditional methods is very time-consuming, since the simulator has to work at the highest level of time resolution.

3.1 Hierarchical Simulation

A general hierarchical simulation scheme is summarized in Figure 4(a). A hierarchical simulation consists of high-resolution, and low-resolution (or coarser), modules. The high-resolution module is the usual discrete-event simulation, while the lower-resolution module consists of one or more of the following components: differential equations (used for example in combat [23] and semiconductor simulations [19]), standard discrete-event simulation, and fluid simulation [25] (currently being developed by our research group and showing promise in computer network simulation). There is an interface between the two modules which forwards some statistics of the high-resolution module output (usually, the average) to the lower-resolution module. The design of the interface is often critical to the success of the entire simulation. Very little attention has been paid to this area: it is one of the main focal points of our research.



(a) Basic Hierarchical Simulation Scheme *(b) Use of Clustering*

Figure 4: Hierarchical Simulation Structure

Hierarchical simulation is a common practice, but the design is always *ad hoc*. A systematic design and analysis framework is definitely needed. In this project, we developed some fundamental components of such a framework. As mentioned above, the key issue in hierarchical simulation is the design of the interface between the hierarchies. Common practice is to use the average as the statistics to be passed to the lower-resolution module.

This is always highly unsatisfactory, since the mean often obscures even the most important features of the high resolution output. In this part of the work, we develop methods to identify suitable techniques to provide higher-resolution output to the lower-resolution module. Our research is based on the theory of random perturbations in dynamic systems (see, e.g. [12, 18]) and the clustering of large vectors using neural networks (see, e.g. [3]).

Impact of Higher Moments. To show why higher moments of the high-resolution module are needed as inputs to the lower-resolution module, let us consider the case of hierarchical combat simulation. This is an important application area and its structure is typical of most hierarchical simulation problems.

In the following, we will present a greatly simplified and idealized treatment of combat simulation. Actual combat simulations are far more complex (e.g., the COSAGE package has 27,000 lines of SIMSCRIPT); however, this simplified version will be a good expository example.

In combat simulation, the high-resolution component is mostly event-driven, while the lower-resolution simulator is typically a generalization of the well-known Lanchester attrition equation [21]. This consists of a set of linear differential equations, stating that the attrition of one side is proportional to the strength of the other side. Incidentally, the Lanchester equations are also used in economics to model competition between large corporations.

The basic Lanchester equations are

$$\dot{b}(t) = -\theta r(t), \qquad \dot{r}(t) = -cb(t) \tag{11}$$

where $b(t), r(t)$ denote the respective strengths at time t of the two sides engaged in combat. Given the initial conditions, $b(0) = b_0$, $r(0) = r_0$ and information on the coefficients θ, c, these equations can be solved.

The coefficients, θ, c, are estimated using a high-resolution discrete-event simulator. By

the very nature of their creation, these estimates have an uncertainty associated with them. It is important to study the effect of this uncertainty on the behavior of the Lanchester equations and ultimately its output, which is the final output of the entire combat simulator.

In the following discussion, purely for brevity of the description, we will assume that c is known perfectly, while θ is a random variable, in fact, however, both θ and c are random [14].

Assume θ is random, with $E[\theta] = \mu$, $\mathrm{Var}[\theta] = \sigma^2$. To understand the impact of the uncertainty (randomness) in θ, we proceed as follows. Let a be a sample value of θ. Then, we can conveniently treat b and r as functions of both a and t; that is, write them as $b(a,t), r(a,t)$. Let $b(a,t)$, $r(a,t)$ satisfy

$$\frac{d}{dt}b(a,t) = -ar(a,t), \qquad \frac{d}{dt}r(a,t) = -cb(a,t) \tag{12}$$

such that $b(a,0) = b_0$, $r(a,0) = r_0$.

¿From these equations we can easily derive differential equations for $\partial b(a,t)/\partial a$ and $\partial r(a,t)/\partial a$ and the second derivatives. These derivatives can determine the second order statistics of the basic quantities $b(\theta,t), r(\theta,t)$, which would be lost in the current "mean value" practice. These derivatives will also provide some refined approximations for $E[b(\theta,t)]$ and $E[r(\theta,t)]$. In particular, if σ is small we would have

$$E[b(\theta,t)] \approx b(\mu,t) + \frac{1}{2}\frac{\partial^2 b(\mu,t)}{\partial a^2}\sigma^2 \tag{13}$$

$$E[r(\theta,t)] \approx r(\mu,t) + \frac{1}{2}\frac{\partial^2 r(\mu,t)}{\partial a^2}\sigma^2 \tag{14}$$

Some immediate observations are:

- The traditional method of using the mean of the high-resolution simulation results as the coefficients of the Lanchester equation could cause serious bias due to the variance.

- Such a bias can be significantly reduced by using high-order statistics.

The above has been an idealized treatment; in actuality, the attrition equations are far more complicated. However, the principle outlined here is clearly generic and holds for more complicated cases as well. Actually, the principal idea is generally applicable to hierarchical simulation in many other areas such as the modeling of manufacturing systems, semiconductor device [19], and telecommunication networks.

We now describe an application in a hierarchical combat simulation model we have worked on. This model, called "Concept Evaluation Model (CEM)", has been used by the U.S. Army Concept Analysis Agency. The high resolution module in this model is called "COmbat SAmple GEnerator (COSAGE)" and it generates battle paths at division or lower levels. The high resolution module contains a set of attrition equations based on the Lanchester principle and is called "ATtition CALculation (ATCAL)". The attrition equations for the direct fire/point fire case are:

$$(\Delta N_k)_{ij} = \overline{N}_i (RATE)_{ij} P_{ijk} [1 - (1 - A_{ijk})\overline{N}_k] \prod_{k'} [1 - A_{ijk'}]\overline{N}_{k'}$$

$$\Delta N_k = (1 - e^{-\Delta N_k / \overline{N}_k}) N_k$$

where:

- $(\Delta N_k)_{ij}$ is attrition of the kth vehicle by the jth weapon of the ith vehicle,
- $\overline{N}_i$, $\overline{N}_k$ are the average number of the ith (red) and the kth (blue) vehicles,
- $(RATE)_{ij}$ denotes the rounds that the jth weapon of the ith vehicle can fire during an engagement,
- P_{ijk} is the probability of kill per round,

- A_{ijk} is the target availability - the fraction of time that the kth target is available for the jth weapon of the ith vehicle,

- k' is the index of vehicles that have higher priority than k.

To implement the perturbation analysis described above we need to calculate derivatives such as $d(\Delta N_k)_{ij})/dP_{ijk}$ from the above equations. The implementation is straightforward.

Clustering. As briefly discussed before, another important issue in presenting the results of the high-resolution module to the low-resolution module is that of aggregating the high-resolution results.

Quite often, the system being simulated is such that the high-resolution simulator produces so widely divergent outputs that it does not make sense to summarize its output over the entire sample space. In such cases, we must subdivide the sample space into segments, and get the high-resolution simulator to produce an appropriate input to the low-resolution simulator for each such segment. We have dealt, in the previous sections, with the issue of how to generate appropriate statistics for each segment. Here, we will consider how to carry out the subdivision of the sample space. Essentially, the low-resolution simulation will be broken down into a number of distinct simulations, one for each segment of the sample space, as depicted in Figure 1(b).

To carry out such a segmentation, the high-resolution paths need first to be grouped by their common features. These features then determine the corresponding low-resolution model. Each high-resolution output group then feeds a corresponding low-resolution simulator.

To group the high-resolution sample paths, we need to perform clustering analysis for usually huge dimensional data. The exact clustering is very time-consuming. The practice of classifying objects according to perceived similarities is the basis for much of science and

engineering. Organizing data into sensible groupings is one of the most fundamental modes of understanding and learning. Clustering methods have been widely applied in pattern recognition, image processing, and artificial intelligence. In this project, we dealt with clustering methods for the preservation of statistics in hierarchical simulation.

A large collection of clustering algorithms is available in a variety of scientific disciplines and new clustering programs continue to appear in the scientific literature. Our focus has been on *Adaptive Resonance Theory* (ART) to cluster high-dimensional data vectors. Advantages of this approach include its computational efficiency, as well as allowing the user to easily control the degree of similarity of patterns placed on the same cluster. Our experimental results corroborate these observations.

ART neural networks were developed by Carpenter and Grossberg [3] to understand the clustering function of the human visual system. They are based on a competitive learning scheme and are designed to deal with the stability/plasticity dilemma in clustering and general learning. It is clear that too much stability would lead to a "stubborn" mind, while too much plasticity would lead to unstable learning. ART neural networks successfully resolve this dilemma by matching the input pattern with the prototypes. If the matching is not adequate, a new prototype is created. In this way, previously learned memories are not eroded by new learning. In addition, the ART neural network implements a feedback mechanism during learning to enhance stability.

Our experiments of using ART neural networks with combat simulation paths have been quite successful [6]. We believe further improvement with the ART structure can lead to a fundamental breakthrough in large data clustering, which is needed in complex systems modeling. A neural network could be developed into a generic numerical clustering tool for many important problems in intelligent data analysis.

It is worth mentioning that during this project we located an alternative non-parametric

clustering method recently been introduced by Domany and co-workers [2] which holds promise for application to large databases. The idea is to embed the data in a spin model with each data point represented by a single spin. The distance between nearby points determines the strength of a ferromagnetic coupling between the spins. A procedure akin to simulated annealing is then applied to this spin system. Using standard Monte Carlo methods for equilibrating spin systems at some temperature, the temperature is lowered to a range where data points with sufficient similarity are clustered. By adjusting the temperature in the "superparamagnetic" regime, coarse or fine-grained clustering can be achieved. It remains an open issue to identify the possible application of this scheme to general data clustering and specific tasks involved in combat simulation.

An Application to a "real-world" complex system. During the course of the project, we encountered an interesting opportunity to test the clustering techniques we developed in the case of a complex manufacturing system. In particular, in working with a large metal manufacturer we were faced with the issue of supplying a low-resolution model of a large plant with the necessary parameters for running it, much like the coefficients of the Lanchester equation in (11). These parameters are to be obtained from detailed (high-resolution) models of the process plans (or flowpaths) for over 10,000 products manufactured in the plant. A flowpath is a specific sequence of Production Centers (PCs) with different processing characteristics at each PC (there are over 100 such PCs). Thus, each flowpath may be thought of as corresponding to a unique product; however, since the low-resolution model cannot possibly handle input data for over 10,000 flowpaths, the objective is to group products with similar flowpaths. For purposes such as forecasting, capacity planning, and lead-time estimation (among others) it is in fact indispensable to have such product groups available: not only it is conceptually infeasible to work with over 10,000 distinct products, it is also practically impossible to input such high-dimensional data for over 10,000 products and 100 PCs into modeling and decision support tools. Moreover, even if there were an

automated way to accomplish this, it would be unrealistic to expect anyone to manipulate or interpret output data with information such as inventory levels and lead times for many thousands of distinct products.

In the effort to establish groups (or clusters) of products based on similarities in flowpaths and processing characteristics, an initial project was set up with plant experts given the task to "manually" create such groupings. The project was quickly abandoned: in addition to the sheer product volume which makes this task prohibitive, it is also difficult to rationally quantify "similarities" in flowpaths and processing data without some systematic means of doing so. We were able to accomplish this task using the clustering techniques we have developed and obtained a "compression" of over 10,000 products to 25-100 product clusters (depending on the aggregation accuracy required, which is completely controlled by the analyst). Of particular interest is the fact that the plant experts who reviewed the results we obtained found "by hindsight" the clusters defined by our method consistent with their expectations.

As mentioned above, it should be possible to evaluate the result of a particular grouping (or clustering) of flowpaths and to compare it to alternative groupings so as to determine the appropriate level of flowpath aggregation desired depending on the application of interest. For some tasks, less than 100 groups may be amply adequate, while for others it may be necessary to use a form of grouping based on tighter similarity requirements that would yield several hundred flowpath clusters aggregated into product groups. In general, the "tighter" the similarity requirements imposed, the larger the number of resulting clusters is likely to be. This capability is an integral part of the clustering tools we are developing and is captured by the so-called "vigilance parameter".

3.2 Basic Concepts of Clustering

In this section, we report our review of some basic concepts in data clustering techniques. This review is important because it not only give the background information for our work, it also motivates ideas and concepts for possible new techniques.

3.2.1 Classification and Clustering

Classification is the actual or ideal arrangement of patterns which are alike, and the separation of those which are not; the purpose of the arrangement is to shape and keep knowledge, to analyze the structure of phenomena, and to relate different aspects of a phenomenon. Applications of classification in science include Library Science and Information Retrieval, Mathematics, Biology, Physics and Chemistry, Social and Political Science.

Clustering is the mathematical technique designed to reveal classification structures in data collected from real-world phenomena; the purpose of clustering is to (a) analyze the structure of the data, (b) relate different aspects of the data to each other, and (c) assist in classification design.

A typical problem is of the following form: Given a set of entities, determine its subsets (called *clusters*), which are homogeneous and well separated. Homogeneity means that entities in the same cluster should resemble each other; separation means that entities in different clusters should not.

3.2.2 Clustering Framework

The basic framework of clustering consists of eight elements:

1. *Sampling:* Select a set $O = \{O_1, O_2, ..., O_N\}$ of N entites.

2. *Data collection::* Observe or measure p characteristics on each of the entities of O. This leads to a data matrix X of size $N \times p$ with binary entries for observations and real entries for measurements.

3. *Dissimilarity:* Construct a matrix $D = (d_{kl})$ of size $N \times N$, which is generated from data matrix X. Dissimilarities satisfy the following properties:

   - Symmetry: $d_{kl} = d_{lk}$
   - Non-negativity: $d_{kl} \geq 0$
   - Vanishing diagonal elements: $d_{kk} = 0$, for $l, k = 1, 2, ..., N$.

   The Euclidean distance on a Euclidean plane is an example.

4. *Types of clustering:*

   We list some of the most important clustering.

   1) Subpartition $SP_M = \{C_1, C_2, ..., C_M\}$ of O with M clusters:$C_j \subset O, C_j \neq \emptyset, C_i \cap C_j = \emptyset$ for $i, j = 1, 2, ..., M$.

   Note that here the clusters are not required to cover the entire set O;

   2) Partition $P_M = \{C_1, C_2, ..., C_M\}$ of O into M clusters: $C_j \neq \emptyset, C_i \cap C_j = \emptyset, \cup_{j=1}^{M} C_j = O$; for $i, j = 1, 2, ..., M$.

   Note that here the clusters are required to cover the entire set O;

   3) Covering $CO_M = \{C_1, C_2, ..., C_M\}$ of O by M clusters: $C_j \neq \emptyset, \cup_{j=1}^{M} C_j = O$; for $j = 1, 2, ..., M$;

   4) Hierarchy $H_{SP} = \{SP_1, SP_2, ..., SP_K\}$ of subpartition of O: Set of K subpartitions $SP_1, SP_2, ..., SP_K$ of O such that $C_i \in SP_k, C_j \in SP_l$ and $k > l \Rightarrow C_j \subset C_i or C_i \cap C_j = \emptyset$.

   Note that here the clusters are not required to cover the entire set O;

5) Hierarchy $H = \{P_1, P_2, ..., P_N\}$ of partition of O: Set of N partitions $P_1, P_2, ..., P_N$ of O such that $C_i \in P_K, C_j \in P_l$ and $k > l \Rightarrow C_i \subset C_j$ or $C_i \cap C_j = \emptyset$.

Note that here the clusters are required to cover the entire set O.

5. *Criterion:* Select a criterion to evaluate the clusterings of the type decided upon in Step 4. Such a criterion may be of one of the following types:

   1) Threshold-type criteria, in which a single dissimilarity determines this value.

   2) Sum-type criteria, in which a sum of dissimilarities involving one entity determines this value.

   3) Sum-sum-type criteria, in which all dissimilarities between pairs of entities of the cluster are used to determine this value.

6. *Algorithm:* Choose or devise an algorithm for the problem defined in Steps 4 and 5.

7. *Computation:* Determine the clusterings of O which optimize the chosen criterion, with the algorithm of Step 6.

8. *Interpretation of results;* Try to use descriptive statistics to summarize the characteristics of each cluster.

We now turn our attention to the description of a class of concrete clustering algorithms referred to as "sequential clustering."

3.2.3 Sequential Clustering

The principle of sequential clustering is as follows. Most commonly used paradigms in cluster analysis, such as hierarchical clustering and partitioning, imply that all entities should be assigned to clusters. However, in most cases, this is unnatural: some entities fit poorly into any clusters, or they are just noise. Thus, sequential clustering searches within the data

(whatever structure there is and nothing more), isolates clearly apparent clusters one at a time, and stops when this cannot be done anymore. A similar procedure is often used in image processing where objects are recognized one after another.

Following the basic framework of the previous section, the sequential clustering scheme works as described next.

Step 1: Sample. Select entities among which clusters are to be found.

Step 2: Data. Measure characteristics of the entities and obtain the sample data matrix X.

Step 3: Dissimilarities. Based on the sample data matrix X ,compute dissimilarities between pairs of entities, and obtain the dissimilarities matrix $D = (d_{ij})$.

Step 4: Criterion. Choose a criterion to evaluate the homogeneity and separation of the clusters to be obtained.

Step 5: Choosing K entities among N. According to the criterion of Step 4, determine a best subset of K entities of O as an optimal cluster C^*(usually K is considered as a parameter).

Step 6: Significance test. Apply formal or informal tests to evaluate if the cluster C^* found in Step 5 corresponds to some part of the structure inherent to O, or only to noise. In the former case, record the list of entities of C^*, remove them from O and return to Step 5; in the latter case proceed to Step 7.

Step 7: Interpretation of results. Describe the clusters found sequentially by the lists of their entities and various techniques of descriptive statistics. Proceed to a substantive interpretation of these results.

To complete a seqential algorithm, one must specify how to choose K entities from N entities and which significance test applies. It is also important to choose the most appropriate clustering criterion for the specific application. In Figures 5, 6 and 7 we illustrate

three different criteria in two-dimensional space. These concepts and related issues will be further explored in future research.

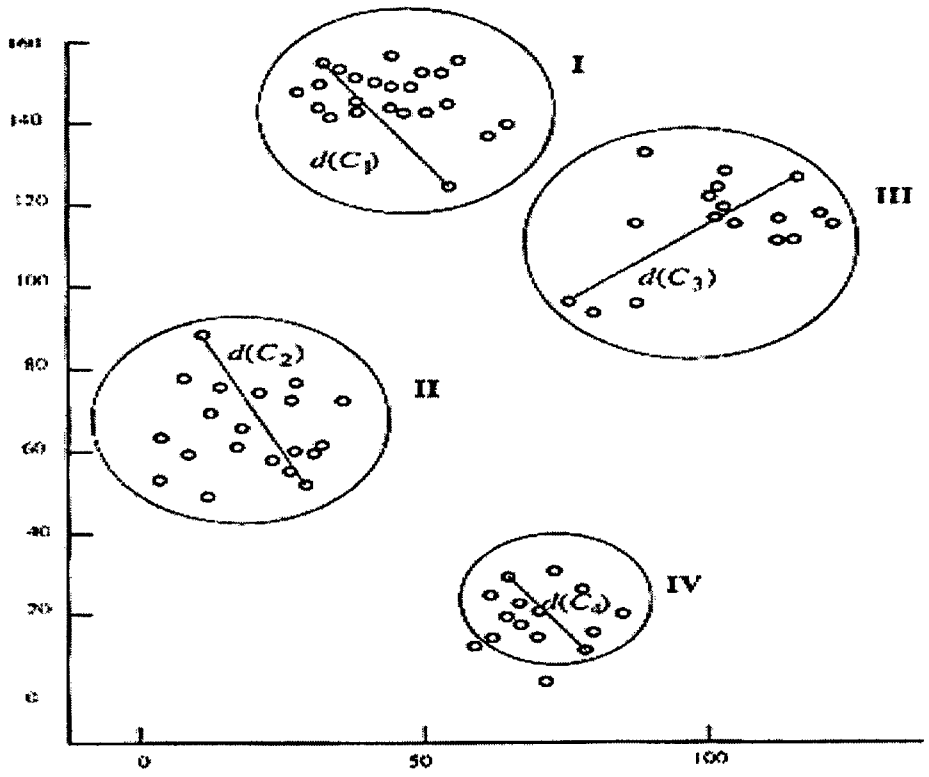


Figure 5: Sequential clustering with minimum diameter criterion.

In summary, clustering is a statistical tool. It aims at extracting a possible cluster structure from a large data set. Based on the selection of clustering type, criterion for clusters and significance test, there are many kinds of clustering algorithms. The choice of the algorithm depends on the specific problem and on computational capacity. Traditional clustering algorithms are tantamount to global optimization for a selected objective function (criterion). They often imply huge computational complexity and are NP-hard. As the need for clustering is increasing, especially for large-dimension data sets, more efficient approaches are required. Some new approaches, such as those based on neural networks and probabilistic methods, have emerged and have shown to be efficient for some problems. In the next section we review the basic principles of an approach based on Adaptive Resonance Theory (ART)

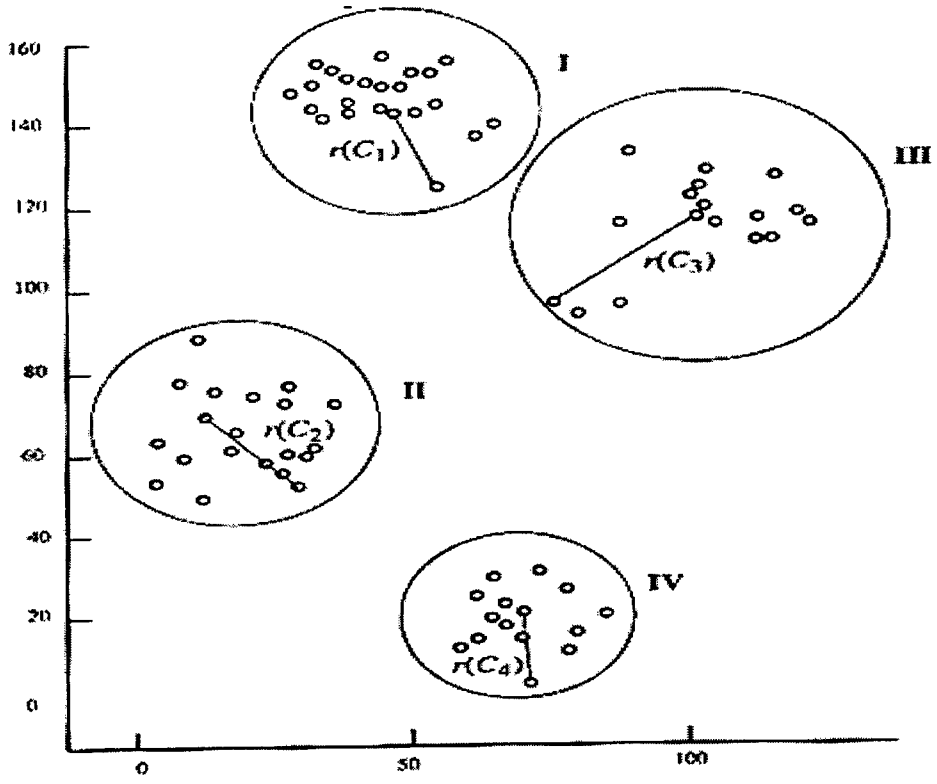


Figure 6: Sequential clustering with minimum radius criterion

neural networks, and present concrete algorithms we have used in this project.

3.2.4 Clustering using Adaptive Resonance Theory (ART)

As we mentioned in the Introduction, this part of our work is motivated by our Path Bundle Grouping approach in hierarchical combat simulation. In dealing with hierarchical simulation models one needs to consider grouping the sample paths generated from the high resolution simulators so as to provide appropriate input statistics to the lower resolution simulator. This requires clustering very high dimensional data vectors (the sample paths from the high resolution simulator). Classical clustering algorithms are not efficient for this purpose. We have used this approach in the Concept Evaluation Model (CEM) of the Concept Analysis Agency in order to group the sample paths from the high resolution Combat Sample

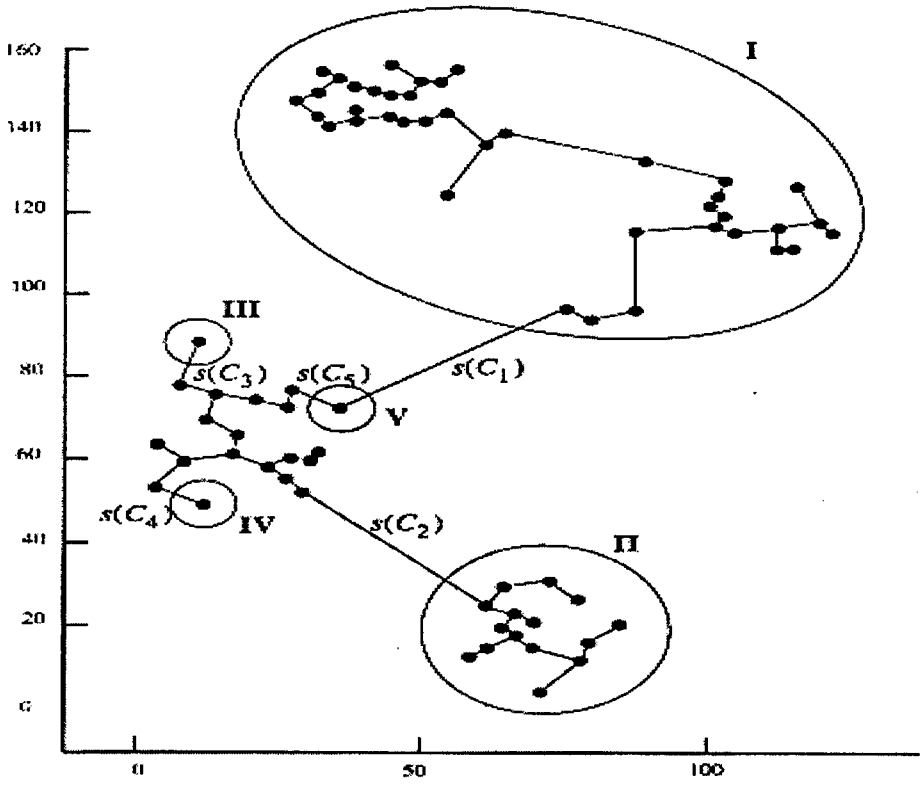


Figure 7: Sequential clustering with maximum split.

Generator (COSAGE) and generate the input to the lower resolution Attrition Calculation (ATCAL). Concrete numerical results are reported in [17].

The common algorithm used for clustering in the ART framework is closely related to the well-known k-means clustering algorithm. Both use single prototypes to internally represent and dynamically adapt clusters. The k-means algorithm clusters a given set of input patterns into k groups. The parameter k thus specifies the coarseness of the partition. In contrast, ART uses a minimum required similarity between patterns that are grouped within one cluster. The resulting number k of clusters then depends on the distances (in terms of the applied metric) between all input patterns, presented to the network during training cycles. This similarity parameter is called vigilance and is denoted by ρ.

The first step in the ART algorithm is the *preprocessing* stage. It is the creation of an

input pattern as an array with a constant number of m elements. ART requires the same pattern size for all patterns, i.e., the dimension of the input space into which all cluster regions shall be placed. Any of the already formed prototypes is of the same dimension m. In addition, the elements of an input pattern must fit constraints concerning, for example, value bounds or the geometric length of the array viewed as a vector. These constraints are characteristics of the different types of ART networks and are needed to make the input comparable to the cluster prototypes. Once the input pattern is formed, it is compared to the n stored prototypes in a *search* stage. If the degree of similarity between the current input pattern and the best fitting prototype J is at least as high as a given vigilance ρ, this prototype is chosen to represent the cluster containing the input. The degree of similarity is typically limited to the range $[0,1]$. If the similarity between input pattern and best fitting prototype does not fit into the vigilance interval $[\rho,1]$, then a new cluster has to be installed, where the current input is most commonly used as the first prototype or "cluster center". Otherwise, if one of the previously committed clusters matches the input pattern well enough, it is adapted by slightly shifting the prototype's values toward the values of the input array.

The primary processing module of the ART network is a competitive learning network, as shown in Figure 8. The m neurons of an input layer F_1 register the values of an input pattern $I = (i_1, i_2, \cdots, i_m)$. Every neuron of an output layer F_2 receives a *bottom-up* net activity t_j, built from all F_1-outputs $S = I$. The vector elements of $T = (t_1, t_2, \cdots, t_n)$can be seen as the result of comparisons between input pattern I and prototypes $W_1 = (w_{11}, \cdots, w_{1m})$,...,$W_n = (w_{n1}, \cdots, w_{nm})$ These prototypes are stored in the synaptic weights of the connections between F_1- andF_2- neurons. Only an F_2- neuron J, receiving the highest net activity t_J, sets its output to 1, while all other output neurons remain 0:

$$u_j = \begin{cases} 1 & \text{if } t_j > \max(t_k : k \neq j) \\ 0 & \text{otherwise} \end{cases} \tag{15}$$

One *possible* way to compute net activities t_j, and by that measure the similarity between I and W_j, is the weighted sum

$$t_j = \sum_{i=1}^{m} w_{ij}\, i_i \tag{16}$$

Variations on this measure are often employed because the value t_j exerts great influence on the resulting cluster. After an F_2-winner J has been found, the corresponding prototype $W_J = (w_{1J}, \cdots, w_{mJ})$ is adapted to the input pattern I. One suitable method for adaptation is to move W_J slightly toward input pattern I as follows:

$$W_J^{(\text{new})} = \eta \cdot I + (1 - \eta) \cdot W_j^{(\text{old})} \tag{17}$$

where the constant *learning rate* $\eta \in [0, 1]$ is chosen to prevent prototype W_J from moving too fast and therefore destabilizing the learning process. Prototypes for this kind of competitive learning network can be initialized either with random values or with values of randomly chosen input patterns from the training sequence.

Competitive learning networks of this kind tend toward unstable categorization whenever the distances between single input patterns vary in too wide a range. Additionally, there is no way to control either the number of clusters produced by the network, or the minimum similarity of patterns in one cluster. In ART, this problem is solved by extending the competitive learning network as shown in Figure 9. A second set of connections is added, sending the F_2-output **U** back to layer F_1. The synaptic *top-down* weights W_{ij} of these connections are, except for a possible scaling factor, identical to the *bottom-up* weights W_{ij}. The *top-down* net activity **V** is usually calculated by

$$v_j = \sum_{j=1}^{n} u_j \cdot w_{ji} \tag{18}$$

This leads to

$$V = U \cdot W_{ji} = W_j \tag{19}$$

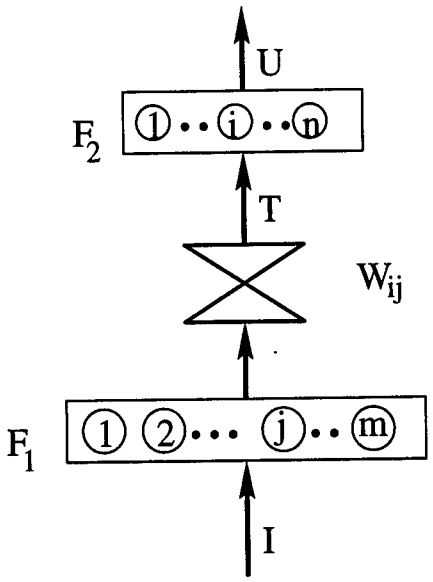


Figure 8: A simplified representation of the competitive learning network

because all F_2-outputs, except u_J, are set to 0 [see (1)]. So input layer F_1 receives prototype W_J, representing the current winning cluster J, as net activity. Next, the most complex part of signal processing in ART networks takes place, i.e., matching prototype W_J with input pattern I. This task is completed in ways characteristic to the different types of ART networks and, uses extensions to the internal structure of layer F_1. This yields a single matching value which is compared to the *vigilance* ρ, defining the minimum similarity between an input pattern and the prototype of the cluster it is associated with. If the matching value is smaller than vigilance ρ, the current winning F_2-neuron is removed from the competition by a *reset* signal. The reset signal forces the activation F_2-neuron J to 0 and another F_2-neuron is activated, receiving the highest net activity t_j of all non-reset output neurons. Once a prototype is found that leads to a matching value with input pattern I, at least as high as vigilance ρ, no further reset signal is applied and the network attains *resonance*. The position of the last winning F_2-neuron indicates the final cluster for input

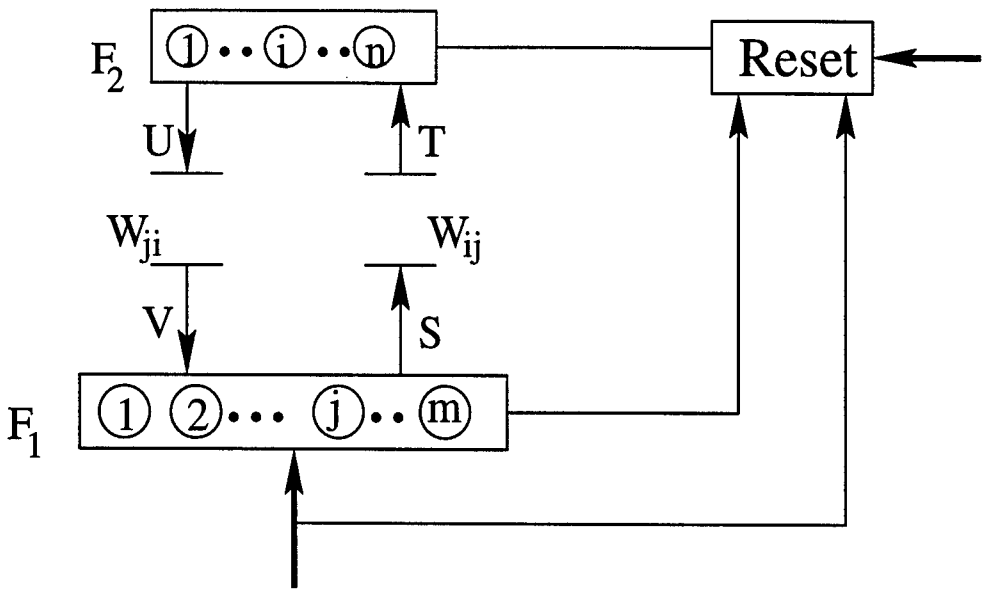


Figure 9: Basic layout of an ART network

I, and the associated prototype is adapted.

Figure 10 demonstrates the similarity concept in the ART2 network. We emphasize that the coordinates may be scaled to provide rich flexibility.

The initial values of prototypes that have not yet been accessed by an input pattern, provide for two key features:

1. Previously accessed prototypes are first compared to the input pattern before an uncommitted prototype is chosen.

2. If none of the committed clusters matches the input pattern well enough, search will end with recruitment of an uncommitted prototype.

The basic structure of an adaptive resonance neural network involves three groups of neurons:

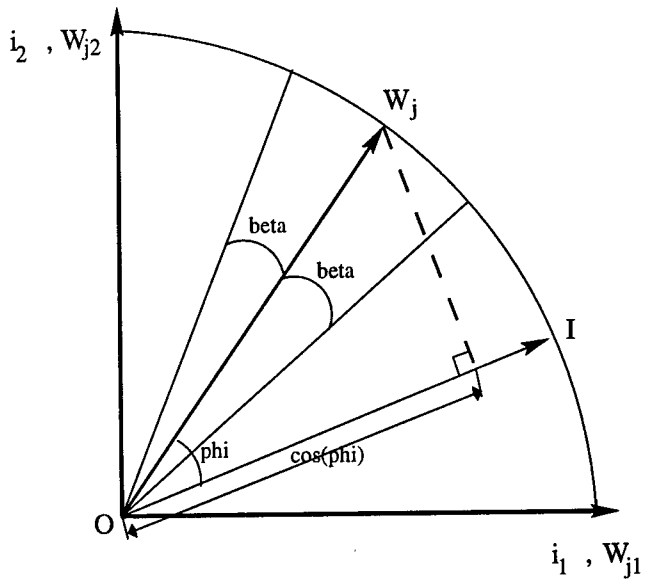


Figure 10: Similarity in ART2 is measured by the angle

- F1 layer: the input processing field
- F2 layer: the cluster units
- Reset mechanism: the mechanism to control the degree of similarity of patterns placed on the same cluster.

The F1 layer has two parts: the *input portion*, denoted by F1(a), and the *interface portion*, denoted by F1(b). In ART2, some processing may occur in the *input portion*. The *interface portion* combines signals from the *input portion* and the F2 layer to compare the similarity of the *input signal* to the *weight vector* for the cluster unit, which has been selected as a candidate for learning.

To control the similarity of patterns placed on the same cluster, there are two sets of connections (each with its own weights) between each unit in the *interface portion* of the

input field and each *cluster unit*:

- The F1(b) layer is connected to the F2 layer by bottom-up weights b_{ij}; each b_{ij} is the weight on the connection from i-th F1 unit to the j-th F2 unit.
- The F2 layer is connected to the F1(b) layer by top-down weights t_{ji}; each t_{ji} is the weight on the connection from j-th F2 unit to the i-th F1 unit.

The F2 layer is a competitive layer: the cluster unit with the largest net input becomes the candidate to learn the input pattern, the activations of all other F2 units are set to 0. Then, the interface units combine information from the input and cluster units.

Whether the cluster unit is allowed to learn the input pattern depends on how similar its *top-down weight vector* is to the *input vector*; the decision is made by the reset unit, based on the signals it receives from the *input* and *interface* portions of the F1 layer. If the cluster unit is not allowed to learn, it is inhibited and a new cluster unit is selected as the candidate. A graphical representation of an ART2 neural network structure is shown in Figure 11, where the operations involved are listed below:

$$
\begin{aligned}
X_i &= \frac{W_i}{e + \|W\|}; \qquad V_i = f(X_i) + bf(Q_i); \\
U_i &= \frac{V_i}{e + \|V\|}; \qquad Q_i = \frac{P_i}{e + \|P\|}; \qquad P_i = U_i + dt_i; \\
Y_j &= \sum_i b_{ij} P_i; \\
f(x) &= \begin{cases} x & x > 9 \\ 0 & \text{otherwise} \end{cases}
\end{aligned}
$$

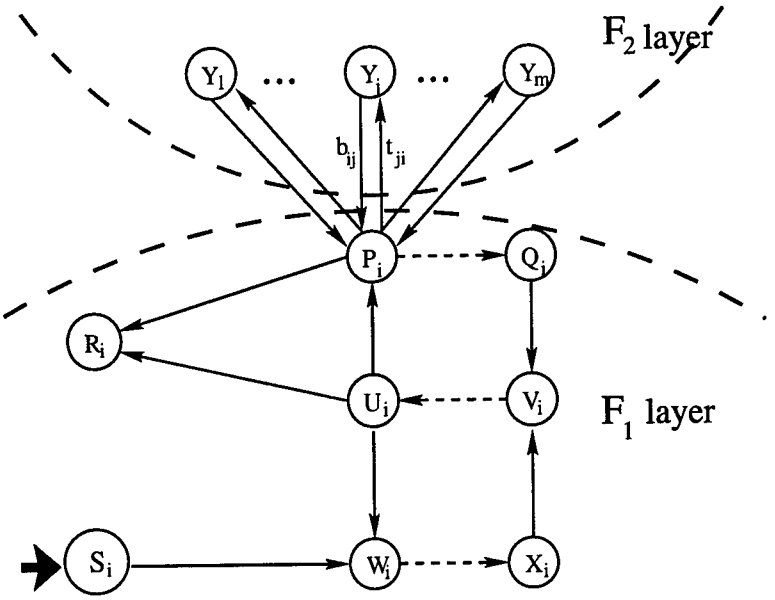


Figure 11: Structure of a typical ART2 neural network

4 METAMODELING USING NEURAL NETWORKS

The purpose of "metamodeling" is to capture complex input/output relationships embodied in a simulator so as to obtain simulation data quickly and efficiently. It is well known that multilayer neural networks are universal function approximators [9]. As such, they are often used for non-parametric modeling, and are well suited to the task of metamodeling, as detailed in our previous report [5]. To briefly summarize the discussion that may be found in Section 3 of [5], a neural network is a device (i.e., software) which conceptually consists of many "neurons" with "weights" attached to them. By adjusting the weights, a neural network is capable of generating response surfaces (i.e., outputs) of extreme generality. The idea of using "neurons" comes from the "stimulus-response" paradigm that our human brain neurons seem to conform to. The metamodeling procedure we have pursued consiusts of the following basic steps.

First, a large-scale simulation with possibly hundreds or thousands of inputs and hundreds or thousands of outputs is executed. The neural network is a device that we have separately

designed (completely independently of the simulator) which is fed by the exact same inputs and outputs of the simulator; it does not, however, intrude in any way into it. While the simulator is running, the neural network observes the inputs and outputs and it "learns" from them. This is called "training" the network. One can visualize the neural network as an "entity" which is highly intelligent but has no knowledge of anything initially to apply its intelligence to. As it observes the simulation unfold, however, it learns from the basic cause-effect (i.e., input-output) relationships it observes. In fact, all the neural network does is adjust its weights so as to emulate the behavior it observes as closely as possible.

When the training is done, the simulator can be taken away. The neural network is now the surrogate model: we may give it some inputs (as if we were giving them to the simulator) and it immediately gives us an output (as the simulator would). So, we can think of it as a "function" which responds to any input by providing some output, except that there is no explicit mathematical expression or formula - just a device (a software routine) that acts as the model.

The main advantages of a neural network were discussed in [5]. It was also pointed out that in order to take full advantage of such an approach, one must design the neural network appropriately and develop efficient ways to accomplish the all-important training process. In [5], we also introduced the *Cascade Correlation Neural Network* (CCNN) as a type of multilayer neural network that builds itself while it learns [10, 15, 20]. The CCNN starts small and makes itself larger during training which usually leads to faster learning and better performance.

One of the objectives of this project was to pursue the study of benchmark problems and evaluate the effectiveness of metamodeling using neural networks. In our earlier work, we concentrated on the Tactical Electronic Reconnaissance Simulation (TERSM) model [22]. TERSM has been extensively studied and was used by previous researchers to develop and

evaluate polynomial metamodels [8, 27, 26] that we were able to compare with our proposed CCNN metamodeling approach extensively in [5].

While TERSM has provided an excellent testbed to study the feasibility of the CCNN metamodeling approach, it lacks some of the features that constitute real challenges to metamodeling. For one, TERSM lacks significant randomness. As provided to us, TERSM is deterministic. For another, TERSM does not exhibit asymptotic relationships which are very common in practice. Exposing such asymptotic behaviors often requires very long simulation runs. Avoiding long simulation runs is one of the real benefits of metamodeling. Additionally, asymptotic behaviors can be very difficult for polynomials to capture, and provide one of the primary motivations for the use of more powerful metamodeling methods like the CCNN we are investigating. With this motivation in mind, we introduced in [5] a new testbed system we called the *Aircraft Refueling and Maintenance System* (ARMS). This system can be viewed as a high-resolution component of a combat simulation model whose output is used by a lower-resolution model. In Section 5.1, we review the ARMS model. In Section 5.2, we present results from a simulation study we have performed, and in Section 5.3, we include the results from our CCNN metamodeling effort applied to the ARMS benchmark problem.

4.1 The Aircraft Refueling and Maintenance System (ARMS)

The basic ARMS model is shown in Figure 12. As illustrated, ARMS is a multiclass queueing system. Jobs from each class $n = 1, \ldots, N$ arrive with average rates λ_n to separate arrival queues with capacities C_n (possibly infinite). The system has θ tokens. At block 1, the jobs compete for tokens on a priority basis, with the class 1 jobs having the highest priority. The job waiting in the highest priority arrival queue will be the first to get a token when one becomes available. After getting a token, the jobs enter a service queue with capacity C_s.

At block 2 the jobs are selected from the service queue according to some service discipline (e.g., first in first out (FIFO)) and routed to one of $k = 1, \ldots, K$ servers. The time to service a job is a random variable $\mu(n, k)$ which can vary as a function of the job class n and the particular server k. Upon completing service, the jobs proceed to block 3, where the token is returned to a token pool, and the job leaves the system.

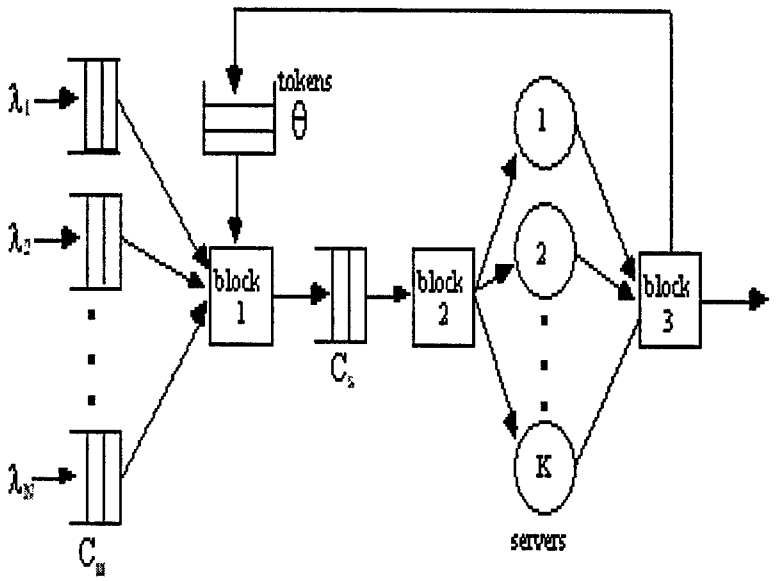


Figure 12: The basic ARMS queueing model.

The basic ARMS queueing model is very general and can be used to represent a large variety of Air Force C^{4}I operations. For example, such a model can be used to represent computer networks, communications systems, or logistics problems. The specific problem we will consider is the aircraft refueling and maintenance system (ARMS) shown in Figure 13. The ARMS problem has aircraft requesting to land at a particular site (airport or aircraft carrier) for refueling and/or maintenance purposes. Depending on aircraft type, a priority is assigned to each aircraft so that high-priority ones are served first. Since landing capacity and associated maintenance resources are limited, a specific number of "permits" (i.e., tokens) are available. An aircraft is, therefore, forced to wait until it receives a permit. Upon receiving a permit, the aircraft is guided to a refueling/maintenance area. If the resources required to

complete the refueling/maintenance process are not immediately available (e.g., personnel, tools, spare parts, fuel), the aircraft is further delayed. When the aircraft completes service, the permit is returned to the permit pool, and the aircraft proceeds to take off and return to action. In studying this system, one is interested in minimizing the expected "down time" of an aircraft, with more emphasis given to certain types of aircraft (the ones given higher priority). At the same time, one is interested in keeping service costs within acceptable levels. From a modeling standpoint, one must therefore determine functional relationships such as the expected down time of a priority 1 aircraft with respect to factors such as the number of permits; or the number of maintenance resources allocated to the refueling/maintenance process.

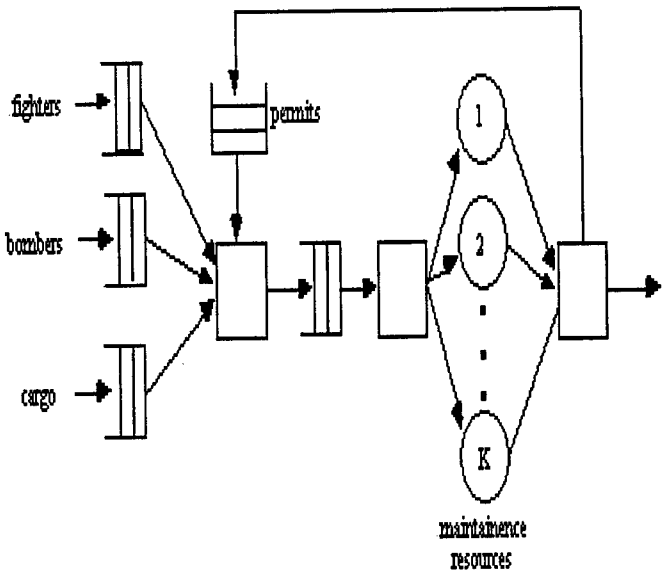


Figure 13: The aircraft refueling and maintenance system (ARMS).

A preliminary analysis focused on the 3-class, single server ARMS model shown in Figure 14 was reported in [5]. In this model, the class 1 jobs have the highest priority, and the class 3 jobs the lowest. Job arrivals are assumed Poisson with rates λ_n, where n is the customer class. In order for a job to be served, it must have one of θ tokens. Jobs with tokens queue up to be served by a single server. At the completion of service, the jobs leave

the system, and the tokens are returned to the token pool for reuse. When different classes of jobs are competing for tokens, the class with the highest priority gets one first. Jobs in the same class compete for tokens on a first come, first served (FCFS) basis. The arrival queues have capacity C_n, and the server queue has capacity $C_s = \theta$. Jobs in the server queue may be served FIFO (with no distinction made between jobs from different classes), or they may be served according to priority (with the highest priority jobs moving to the front of the queue). In any case, service is nonpremptive (once a job begins service, service cannot be interrupted, and will continue until completion), and the service time is an exponential random variable with parameter μ_n.

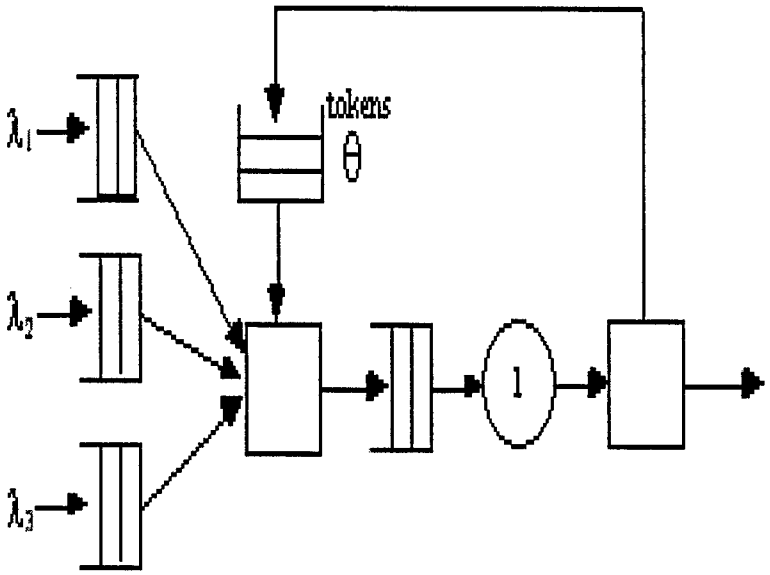


Figure 14: System used for analysis.

4.2 Modeling and Simulation Analysis of ARMS

In order to study the ARMS, i.e., understand its basic dynamic behavior and develop a simulation model for it, a detailed discrete event system (DES) description was developed in [5]. We review it here so as establish the basic notation and terminology required. We begin by defining the state of the ARMS model as follows,

$Q_n \in 0, 1, \ldots, C_n$ - number of class n jobs in the n-th arrival queue

$Q_q \in 0, 1, \ldots, \theta$ - number of tokens in the token pool

$Q_s \in 0, 1, \ldots, \theta$ - number of jobs in the server queue (recall $C_s = \theta$)

where $n \in 1, 2, 3$. State changes in the system are caused by four types of events,

A_n - arrival of a class n job to the n-th arrival queue

A_q - arrival of a token to the token pool

A_s - arrival of a job+token pair to the server queue

D_s - departure of a job+token pair from the server

Job arrival events are always feasible. Token arrival events are only feasible when there are jobs being serviced (i.e., when there are jobs in possession of a token). Server arrival events are only feasible when tokens are available in the token pool. Departure events from the server are only feasible when there a job is being serviced.

The state transition mechanism describing how the state of the system changes in response to the various events is given below.

Job Arrival - A_n

(1) $Q_n > 0, Q_n < C_n \rightarrow Q_n = Q_n + 1$

(2) $Q_n = C_n \rightarrow Q_n = Q_n$, record blocking of a class n job

(3) $Q_n = 0, Q_q = 0 \rightarrow Q_n = 1$

(4) $Q_n = 0, Q_q > 0 \rightarrow Q_q = Q_q - 1$, schedule A_s to occur immediately

(5) $Q_n = 0, Q_q > 0, Q_s = 0 \rightarrow Q_q = Q_q - 1$, schedule D_s to occur in μ_n seconds

In equation (1) a class n job arrives to find its arrival queue not empty, but not at capacity either. The job is added to the back of the queue, where it must wait behind the other jobs in the queue before it can compete for a token. Jobs in the same class compete FCFS with other jobs in the same class for a token. In equation (2) a class n job arrives to find its arrival

queue at capacity. Since the arrival queue is full, the job is blocked, and not allowed to enter the system. Blocking is usually undesirable, and should generally be avoided. In equation (3) a class n job arrives to an empty arrival queue, but finds that no tokens are available. The job, therefore, must wait in its arrival queue for a token to become available. In equation (4) a class n job arrives to an empty arrival queue, and finds a token available (this implies that all other arrival queues must be empty). The job takes the token from the token pool, and proceeds to the server queue. This triggers an A_s event to occur immediately. Finally in equation (5) a class n job arrives to find an empty arrival queue, an available token, and an empty server queue. This job takes a token from the token pool, and immediately begins service. The service time for the job is μ_n, and a service completion event D_s is scheduled to take place in μ_n seconds.

Service Completion - D_s

(1) $Q_s = 0 \rightarrow$ record performance for this job, schedule A_θ

(2) $Q_s > 0 \rightarrow$ record performance for this job, schedule $A_\theta, Q_s = Q_s - 1$, schedule D_s

In equation (1) a job has just completed service and no other jobs are waiting in the server queue for service. The system time (down time) for the job is recorded, the job leaves the system, and the token is returned to the token pool (by scheduling an A_θ event to occur immediately). In equation (2) a job has just completed service, and other jobs are waiting for service in the server queue. As before, the system time for the job that just completed service is recorded, the job leaves the system, and the token is returned to the token pool. In addition, the server queue is decremented, and the job at the front of the server queue begins service. The time to service this job will be μ_n, and a service completion event D_s is scheduled to take place in μ_n seconds. Note, when the server queue is a priority queue, the high priority jobs are shuffled to the front of the queue. Otherwise, jobs are served in the order they arrived to the server queue, independent of their priority.

Server Queue Arrival - A_s

(1) $Q_s = 0 \rightarrow$ schedule D_s

(2) $Q_s > 0 \rightarrow Q_s = Q_s + 1$

In equation (1) a job+token pair arrives to the server queue. Since the server queue is empty, the job immediately begins service. The time to service the job is μ_n, and a service completion event D_s is scheduled to take place in μ_n seconds. In equation (2) a job+token pair arrives to the server queue. In this case, the server queue is not empty, so the job is added to the queue. If the server queue is a priority queue, the job is placed behind the last job in its class. Otherwise, the job is placed at the back of the queue, independent of its priority.

Token Queue Arrival - A_θ

(1) $Q_\theta > 0 \rightarrow Q_\theta = Q_\theta + 1$

(2) $Q_\theta = 0, Q_n > 0 \rightarrow Q_n = Q_n - 1$ (highest priority), schedule A_s

(3) $Q_\theta = 0, Q_n = 0 \rightarrow Q_\theta = 1$

In equation (1) a token is returned to the token pool. Since the token pool is not empty, the token is added to the pool. Note, the token pool will contain tokens only when all arrival queues are empty. In equation (2) a token returning to the token pool finds that the token pool is empty, and that there is a job in at least one of the arrival queues. In this case, the job at the front of the highest priority arrival queue takes the token, leaves its arrival queue, and proceeds to the server queue. In equation (3) a token returning to the token pool finds that the token pool is empty, and that there are no jobs waiting in any of the arrival queues. In this case, the token remains in the token pool.

Based on the DES description above, a simulator was developed, and data were collected to assess the effects that the various parameters have on the system performance. The

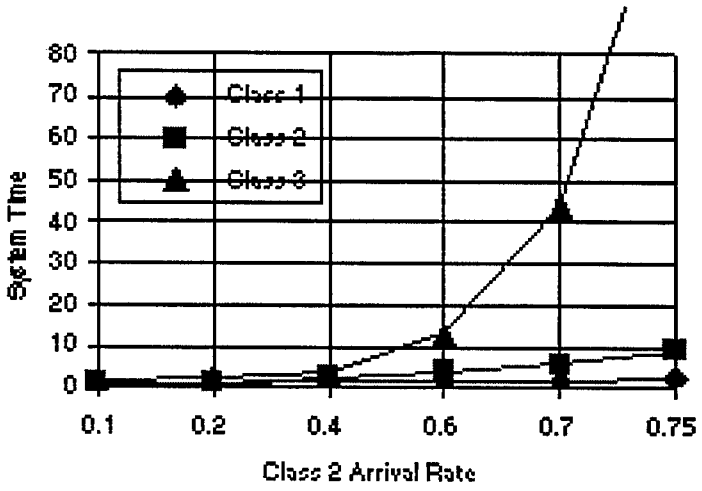


Figure 15: Effect of changing the class 2 arrival rate when the server queue has priorities.

performance measure we use is the *mean system time* for each job class. The system time is the interval from when a job arrives to the system to the time the job completes service and leaves the system. In the ARMS problem, the service time is the "down time" of an aircraft. For the experiments that follow, the arrival queues are assumed to have infinite capacities, i.e., $C_n = \infty$ for $n = 1, 2, 3$.

4.2.1 Arrival Rate Analysis

Some preliminary results for this analysis were reported in [5] and are briefly reviewed here before extending them.

Figure 15 shows what happens when the server queue is a priority queue, which allows the higher priority jobs to jump to the front. In this case, the class 3 jobs really suffer. Not only do the class 3 jobs get starved for tokens, but even when they do get a token, they keep getting pushed to the rear of the server queue. The class 1 jobs, on the other hand, are unaffected by the arrival rate of the class 2 jobs.

Figure 16 shows the effect of changing the arrival rate of the class 3 jobs when the server

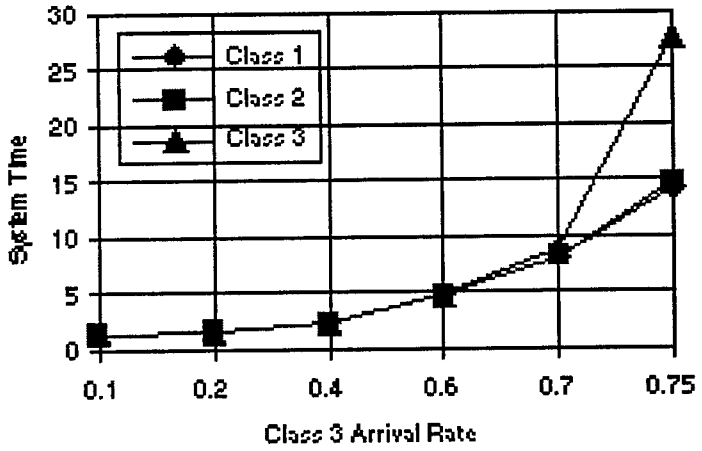


Figure 16: Effect of changing the class 3 arrival rate when the server queue is FIFO.

queue is FIFO. As can be seen, the effects of changing the class 3 arrival rate are very similar to the effects of changing the class 2 arrival rate. As the class 3 arrival rate increases, the system times of the other two classes increase, because they have to wait longer in the server queue. But, again, no matter how high the arrival rate of the class 3 jobs gets, the system times of the class 1 and 2 jobs will never go much higher than about 21 seconds (1 second waiting for a token, 19 seconds waiting in the server queue, and 1 second being served).

Figure 17 shows what happens when the server queue is a priority queue. In this case, the class 1 and 2 jobs are unaffected by the class 3 arrival rate.

To summarize, we have seen that changing the arrival rate of the class 1 jobs has a significant effect on the system times of the other classes. When the server queue is FIFO, with no distinction between classes, we saw that increasing the arrival rate of the lower priority jobs effects the system times of the higher priority jobs by forcing them to wait longer in the server queue. We did notice, however, that, regardless how high the arrival rate of the lower priority jobs becomes, the system times of the high priority jobs never exceeds a certain ceiling that depends on the number of tokens in the system. When the server queue is has priorities, and allows high priority jobs to jump to the front of the queue,

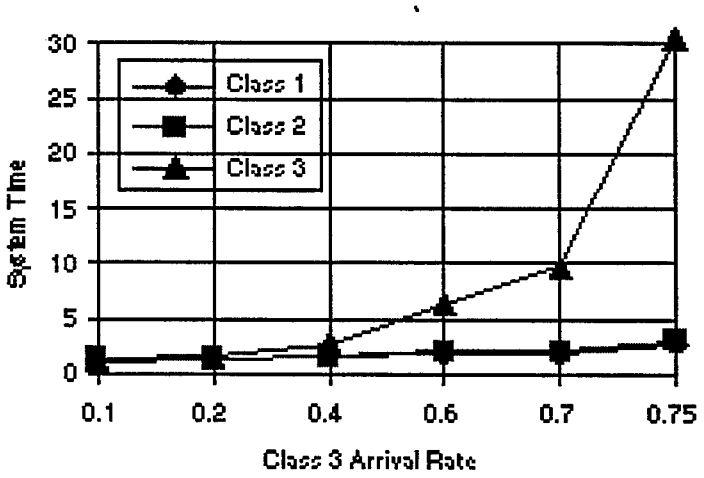


Figure 17: Effect of changing the class 3 arrival rate when the server queue has priorities.

the behavior is different. In this case, the arrival rates of the lower priority jobs have little effect on the system times of the high priority jobs. The low priority jobs, however, are severely effected by increases in the arrival rates of high priority jobs. We also noticed a characteristic which is typical of queueing systems: As the arrival rate increases, the service times asymptotically approach infinity. As a final remark, we note that, as the arrival rates becomes high, very long simulation runs are required to collect statistically reliable data. This is a common feature of queueing systems: When the system is running at or near its stability limits (the point where jobs begin to arrive faster than the server can possibly process them), long simulation runs are needed to collect statistically reliable performance data.

4.2.2 Service Time Analysis.

Here we see how changing the service time of the class n jobs affects the system times of the other classes. As before, we set the number of tokens to $\theta = 20$, so that the token loop can be neglected. We set the arrival rates for each class to be equal at $\lambda_1 = \lambda_2 = \lambda_3 = 0.2$ jobs/second. Then we varied the service time for one class at a time. The service times for

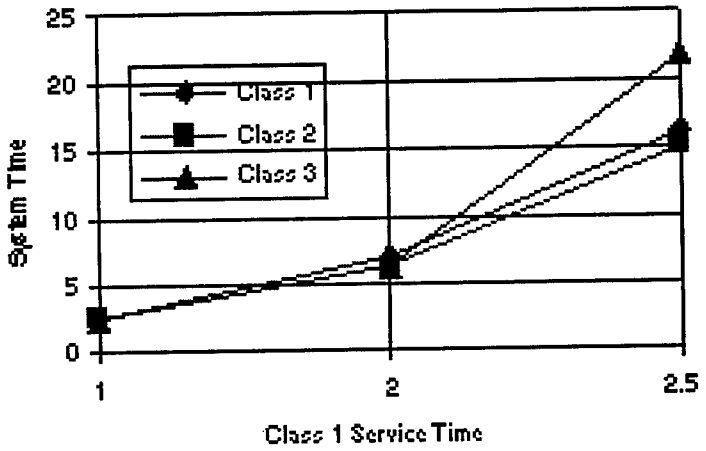


Figure 18: Effect of changing the class 1 service time when the server queue is FIFO.

the classes that were not being varied were set to $\mu = 1$ second. For the experiments in this section, the server queue is FIFO, with no distinction between job classes.

Figure 18 shows what happens when we change the service time of the class 1 jobs. As we see, the effect is very similar to increasing the arrival rate of the class 1 jobs. That is, as the service time of the class 1 jobs increases, the other two classes become starved for tokens, and their system times begin to increase. The mechanism by which this happens is as follows: As the service time increases, the probability that a class 1 job will be waiting when a token becomes available increases. The class 1 jobs take the tokens, and the other two classes are forced to wait. As the system time of the class 1 jobs becomes increasingly larger, the other job classes never get a token, and their arrival queues become unstable. Eventually, the arrival rate of the class 1 jobs exceeds the rate at which the system can process them (because of the long service time), and the class 1 arrival queue will also go unstable. As we saw before, the onset of instability is marked by an asymptotic rise in the system time.

As seen in Figure 19 and Figure 20, the effects of changing the service times of the class 2 and class 3 jobs has very much the same effect as changing their arrival rates.

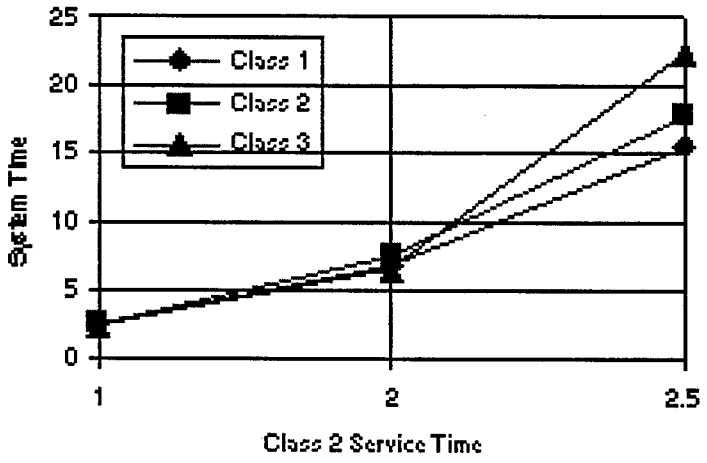


Figure 19: Effect of changing the class 2 service time when the server queue is FIFO.

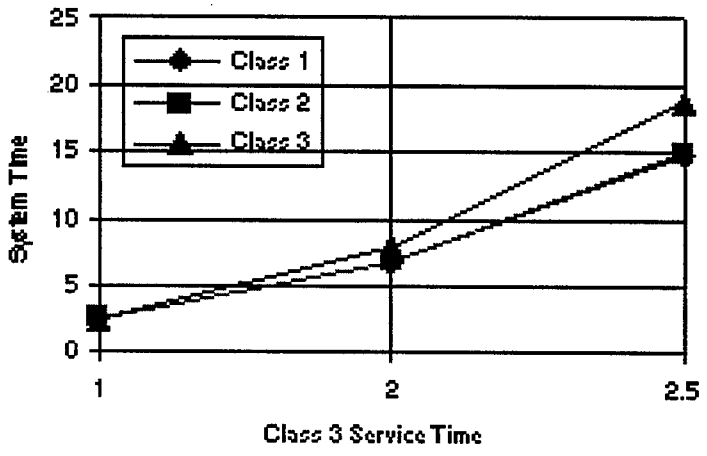


Figure 20: Effect of changing the class 3 service time when the server queue is FIFO.

To summarize, changing the service time has an effect that is very similar to changing the arrival rate. As the service time of the class 1 jobs increases, these jobs utilize most of the tokens, and the other two classes suffer. As the service time of the class 2 jobs increases, they make use of any tokens not used by the class 1 jobs, and the class 3 jobs suffer. As the service time of the class 3 jobs increases, the other two classes must wait behind them in the server queue. As before, if the server queue were a priority queue and not a FIFO queue, the lower priority classes would have much less effect on the higher priority classes.

4.2.3 Token Influence.

Here we look at the effect of changing the number of tokens. To do so, we fix the arrival rates of the three job classes to $\lambda_1 = \lambda_2 = \lambda_3 = 0.30$ jobs/second and their service times to $\mu_1 = \mu_2 = \mu_3 = 1$ second/job. Then we varied the number of tokens.

Figure 21 shows what happens as the number of tokens is decreased from $\theta = 20$ down to $\theta = 1$. In this figure, the server queue is FIFO, with no distinction between classes. What is interesting is that the system times of the class 1 and class 2 jobs actually goes down as the number of tokens is decreased. This happens because, as the number of tokens is decreased, the time that a job must wait in the server queue is decreased. Consequently, the overall system time is decreased. The number of tokens indicates the maximum number of jobs that can be present in the server queue. When jobs in the server queue are served FIFO, more tokens means longer waiting in the server queue. For the class 3 jobs, however, their system times increase as the number of tokens is decreased. This is because the probability that a class 1 or class 2 job will be waiting when a token becomes available increases as the number of tokens decreases, and the class 3 jobs become starved for tokens.

Figure 22 shows what happens when the server queue has priorities. As can be seen, the system times, in this case, are unaffected by the number of tokens. When there are many

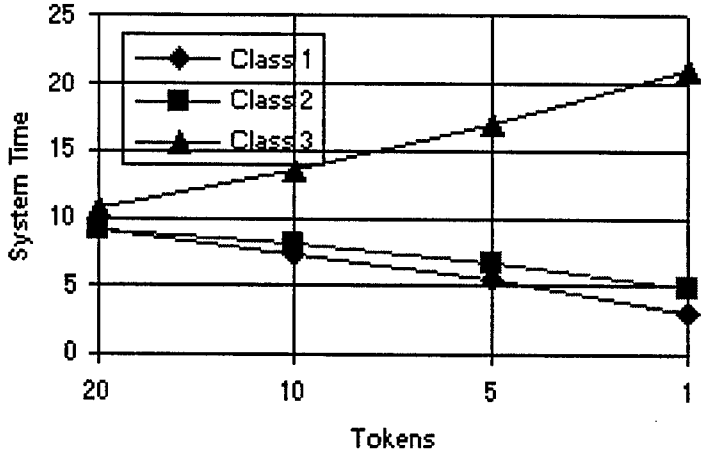


Figure 21: Effect of changing the number of tokens when the server queue is FIFO.

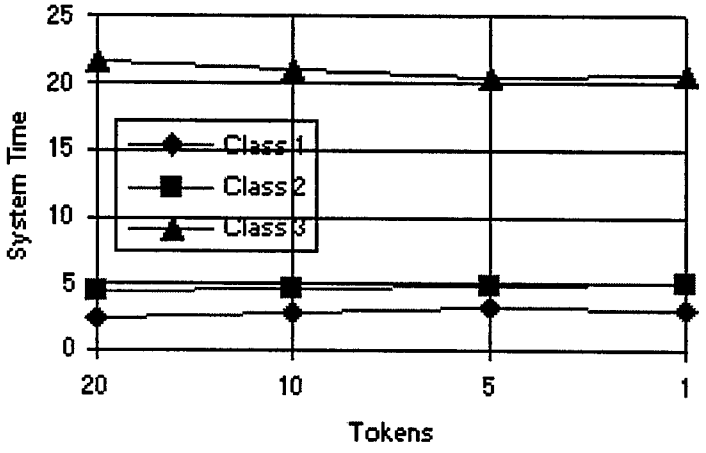


Figure 22: Effect of changing the number of tokens when the server queue has priorities.

Input Variable	Lower Limit	Upper Limit
λ_1 (customers/sec)	0.1	1.0
λ_2	0.1	1.0
λ_3	0.1	1.0
θ	1	20

Table 1: Range of interest for each input variable

tokens, the jobs wait in the server queue. When there are few tokens, the jobs wait in their respective arrival queues. The amount of time they wait, however, does not change. All that changes is where they wait.

4.3 Metamodeling of ARMS using a Cascade Correlation Neural Network

For the purposes of metamodeling, we concentrated on the relationship between four inputs: $\lambda_1, \lambda_2, \lambda_3$ (arrival rates for the three customer classes), and θ (number of tokens); and the output, $J = s_1$, i.e., the service time of the class 1 (highest priority) jobs. The service time is considered to be the time from the job arrival at the system to the time the job finishes service and departs the system. In the context of aircraft refueling and maintenance, this is the "downtime" or amount of time that an aircraft is not available for service. The range of inputs are given in Table 1..

This four-input single-output case was also studied using polynomial metamodels based on the techniques reported in [8, 27, 26] and also used in our earlier work [5].

Some CCNN results on ARMS metamodeling were partly reported in [6] (included in the Appendix). In this section, we have assembled all our numerical results, including some new 3-D plots for the ARMS model.

We have three sets of data:

- Training set for CCNN, size=500, uniformly randomly generated in the input range
- Test set, size=500, uniformly randomly generated in the input range
- Training set for 4th-order polynomial, size=49, generated by layered CCD design [5],[6].

The 4th order polynomial metamodel we used is given in [6] and a square root transformation to the output is used. For CCNN training, we used two schemes:

- CCNN(I): The output is simply rescaled version (here the factor is 0.1) of the real output
- CCNN(II): Natural log transformation to the real output is used.

When the arrival rate approaches the service rate, the output rapidly increases, so it is reasonable to make use of a log transformation which has the property of "flattening" the output curve and making the learning task easier.

Results are shown in Table 2. The CCNN(I) in the table has 25 hidden units; the CCNN(II) has 23 hidden units. Although the training of the CCNN is done according to the mean squared error (MSE), we also look at the mean squared relative error (MSRE) which provides a better picture of the actual metamodel performance. ¿From our earlier work, we know that the ARMS output can explode when the parameters approach a certain region. The MSRE is therefore more appropriate than the MSE in showing how a metamodel performs on the whole range. The MSRE is defined as

$$\frac{1}{n}\sum_{i=1}^{n}(e_i/y_i)^2$$

	training set (size:49)	training set (size:500)	test set (size:500)
Polynomial	1.874 (11.0%)	22.76 (35.0%)	17.74 (34.3%)
CCNN(I)	85.81 (64.2%)	3.438 (13.5%)	13.01 (22.9%)
CCNN(II)	86.69 (27.0%)	6.890 (12.3%)	10.57 (16.3%)

Table 2: CCNN and polynomial metamodeling results of ARMS

where e_i is the error and y_i is the output. The numbers in the Table 2 show the MSE and, in parenthesis, the square root of the MSRE expressed as a percentage value.

It can be seen from Table 2 that the 4th order polynomial performs well on the 49-point training set. However, the MSRE values are quite high for the other two 500-point sets. This is attributed to the fact that most points in the 49-point data set are for extreme cases where certain λ_i values are too large or the number of tokens is too small.

Note that CCNN(I) achieves good training results without too many hidden units. However, it cannot be well generalized to the test data set and performs poorly for the 49-point set due to the choices of extreme values.

The CCNN(II) results show improvement for all data sets, especially when we look at the MSRE criterion. This is due to taking the log of the output, which makes the surface "flatter" as already mentioned.

Next, we present some 3-D plots for the ARMS model in Figures 23 through 31.

Figures 23 through 25 show the case where we fix $\lambda_2 = 0.25, \lambda_3 = 0.3$ and vary λ_1 and θ. This corresponds to a case where class 2 and class 3 jobs have relatively comparable and moderate traffic. We can observe the service time of class 1 for different class 1 arrival rates and different amounts of tokens. From Figure 23, we see that S_1 increases in both dimensions (λ_1 and θ) and goes up drastically on the edge ($0.8 < \lambda_1 < 1$). Figure 24 shows the corresponding CCNN(I) metamodel yielding $MSE = 10.18$ and $\sqrt{MSRE} = 29.1\%$.

Figure 25 is the CCNN(II) result, with $MSE = 10.58$ and $\sqrt{MSRE} = 19.1\%$.

Figures 26 through 28 are for the case where $\lambda_2 = 0.2, \lambda_3 = 0.4$. The shape is not much different from that in the previous setting. The performance for CCNN(I) and CCNN(II) are, $MSE = 9.72$, $\sqrt{MSRE} = 25.0\%$ and $MSE = 8.15$ and $\sqrt{MSRE} = 18.1\%$, respectively.

Figures 29 through 31 show the case when we fix $\lambda_1 = 0.3, \theta = 10$, that is, we observe the influence of lower priority classes on class 1 jobs for a particular class 1 arrival rate and amount of tokens in the system. From Figure 29 we see a surface with much smaller variation in the output. The surface is mostly flat for large values of λ_2 and λ_3, but has a rather sharp slope at the corner for small λ_2, λ_3. Figures 30 and 31 show the results of CCNN(I) and CCNN(II), with $MSE = 1.153$, $\sqrt{MSRE} = 26.9\%$ and $MSE = 0.776$ and $\sqrt{MSRE} = 10.7\%$, respectively. It is interesting to note that CCNN(I) seems to learn the flat shape better than CCNN(II) although the overall performance of the former is not as good as that of the latter.

Observe that the 3-D plots for the simulated data show some very rough surfaces, especially for the first two cases where the output increases significantly. The spikes at the edge are the result of insufficient sample sizes (the stopping condition for all our simulations is defined by the number of jobs served exceeding 1000). Recall that when λ_1 increases, the queueing system will become unstable.

Finally, we performed another series of experiments where the sizes of training sets for both polynomial and CCNN metamodels are increased. Results are given by Table 3. The polynomial is obtained using a 500-point training set; the CCNN(II) with 23 hidden units is trained using a 1000-point data set. The results suggest that additional training points for the polynomial metamodel do not lead to better performance, and the same is true for the CCNN metamodel.

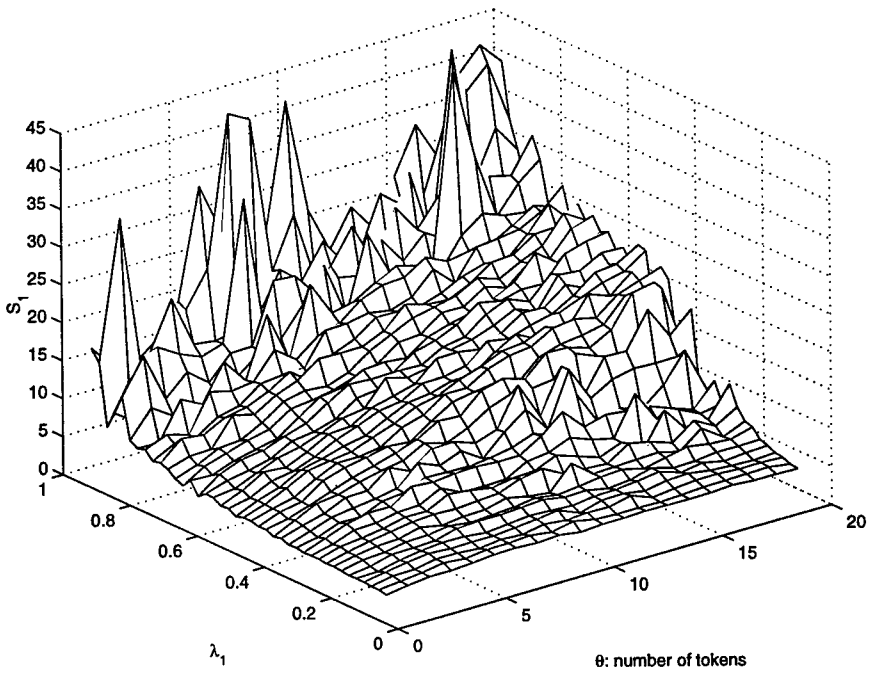


Figure 23: Simulation: Output S_1 vs. (λ_1, θ), $\lambda_2 = 0.25, \lambda_3 = 0.3$

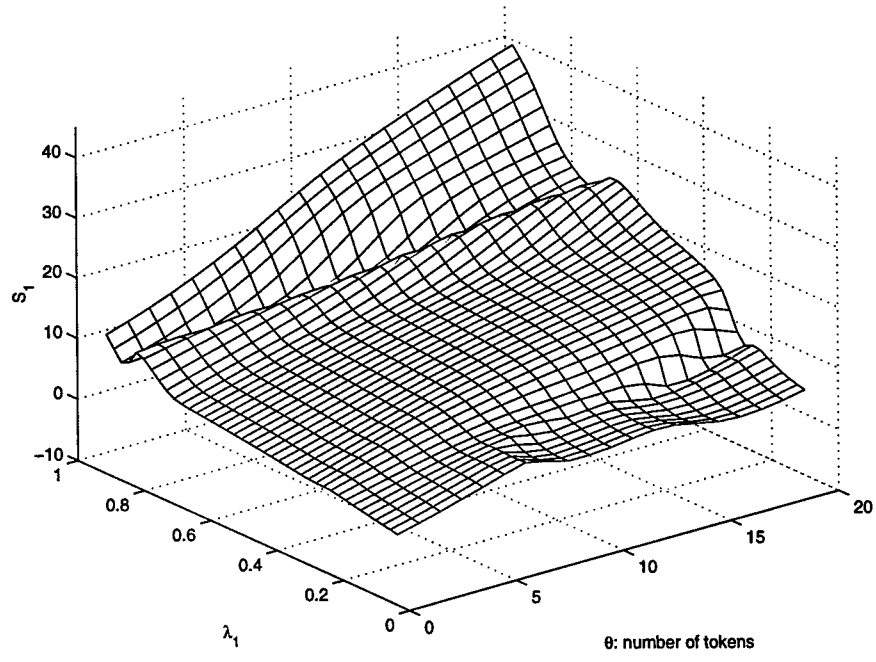


Figure 24: CCNN(I): Output S_1 vs. (λ_1, θ), $\lambda_2 = 0.25, \lambda_3 = 0.3$

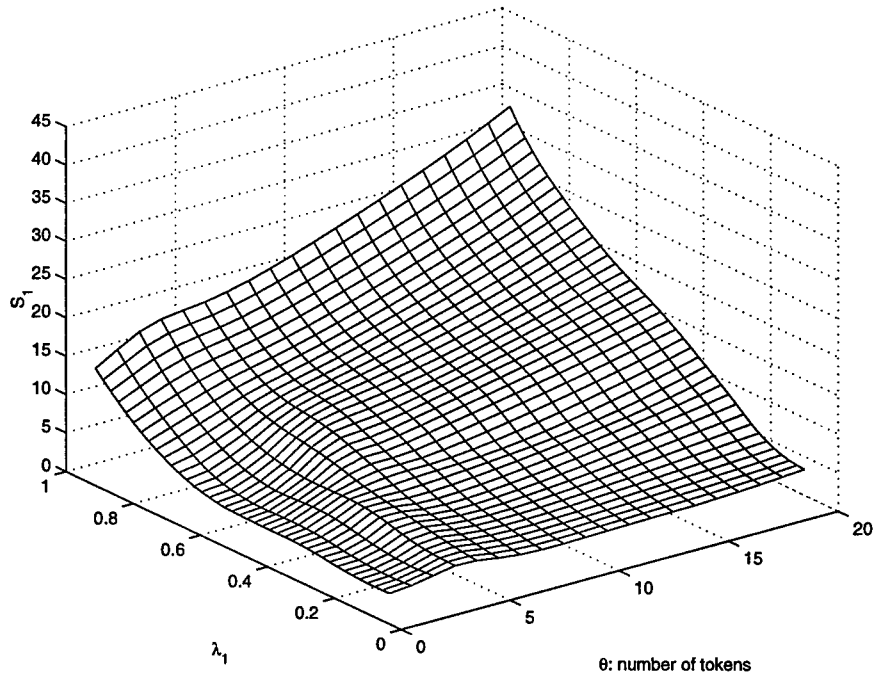


Figure 25: CCNN(II): Output S_1 vs. (λ_1, θ), $\lambda_2 = 0.25, \lambda_3 = 0.3$

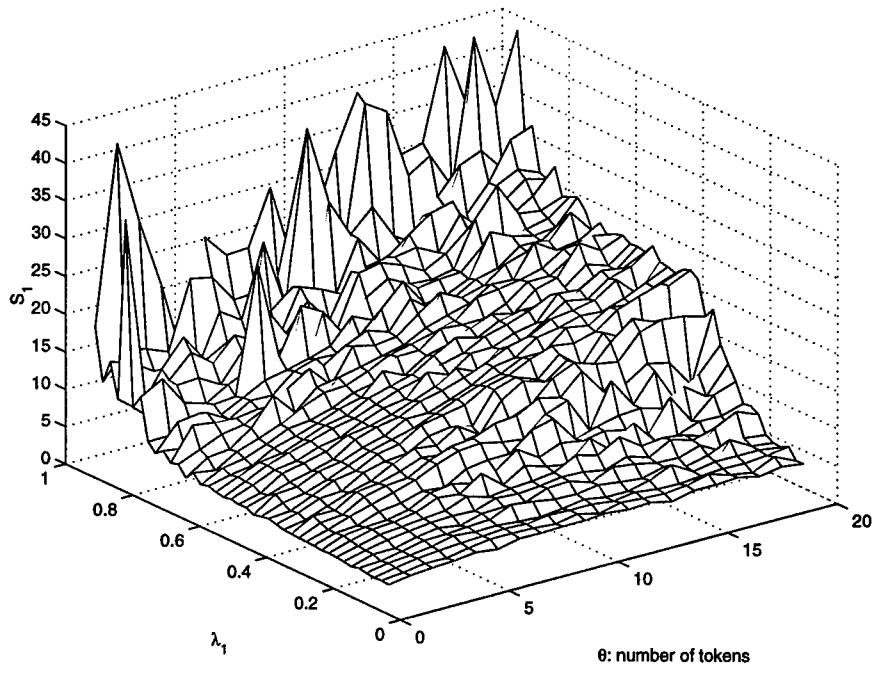


Figure 26: Simulation: Output S_1 vs. (λ_1, θ), $\lambda_2 = 0.2, \lambda_3 = 0.4$

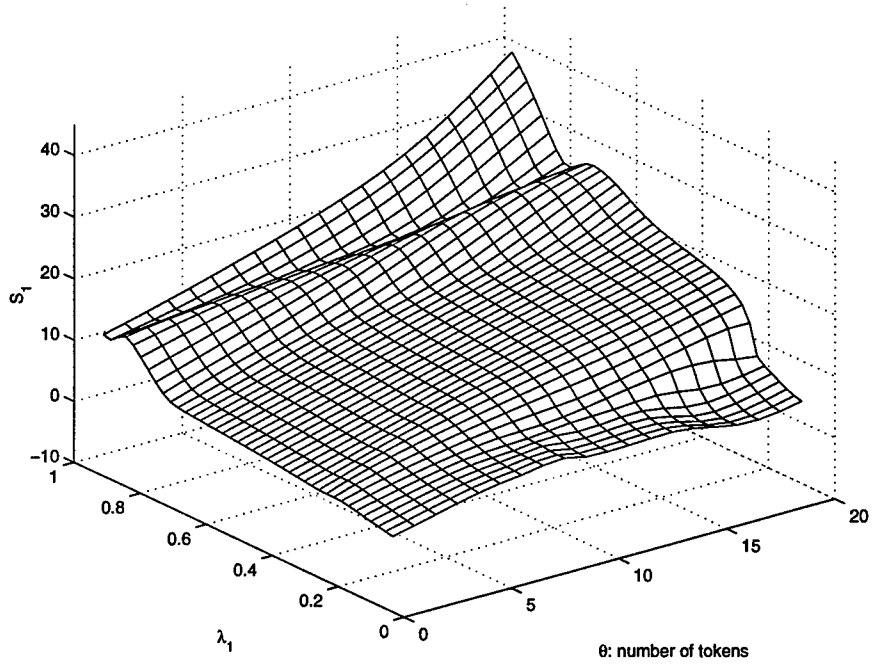


Figure 27: CCNN(I): Output S_1 vs. (λ_1, θ), $\lambda_2 = 0.2, \lambda_3 = 0.4$

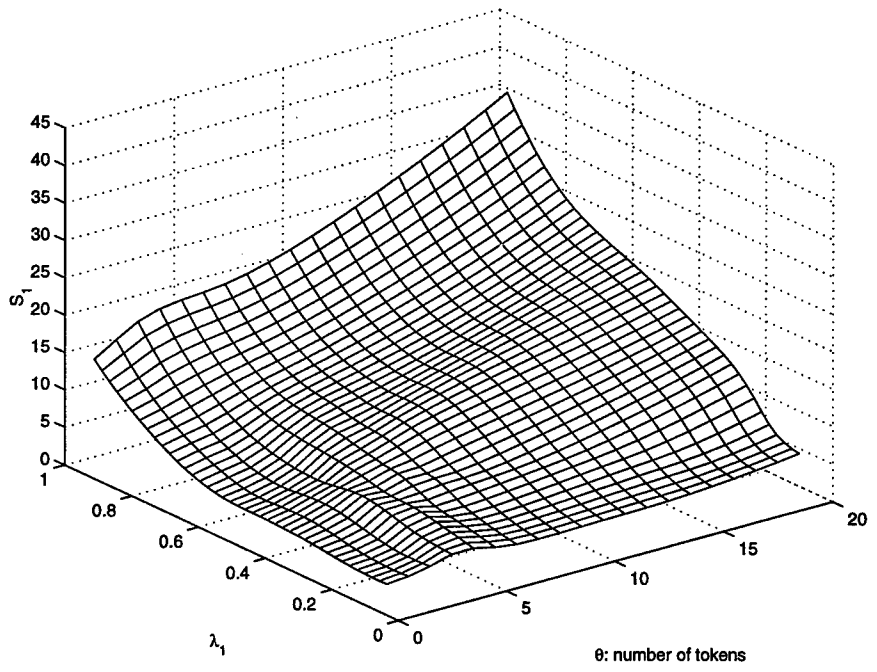


Figure 28: CCNN(II): Output S_1 vs. (λ_1, θ), $\lambda_2 = 0.2, \lambda_3 = 0.4$

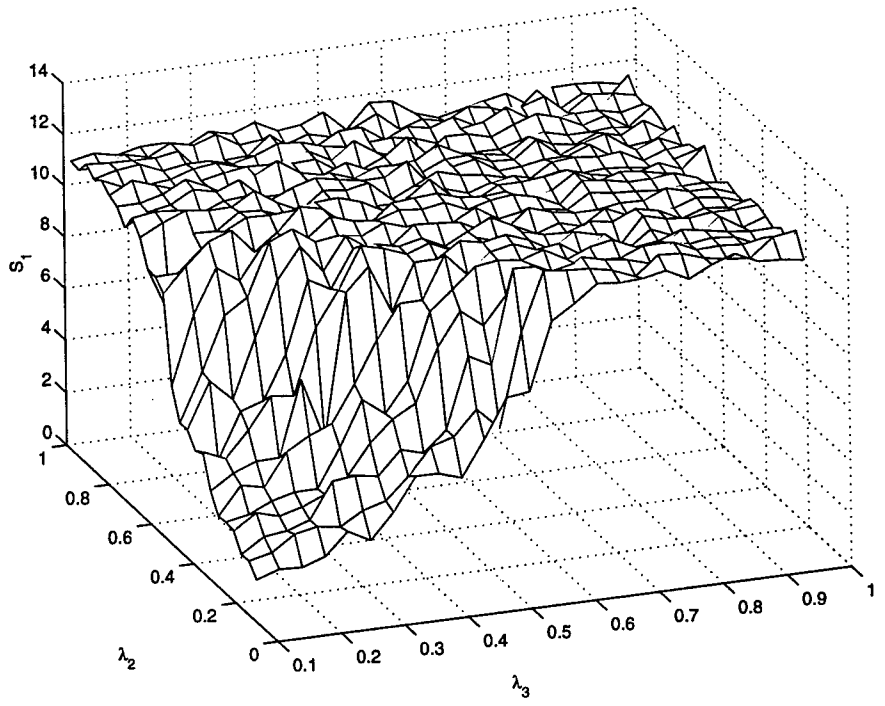


Figure 29: Simulation: Output S_1 vs. (λ_2, λ_3), $\lambda_1 = 0.3, \theta = 10$

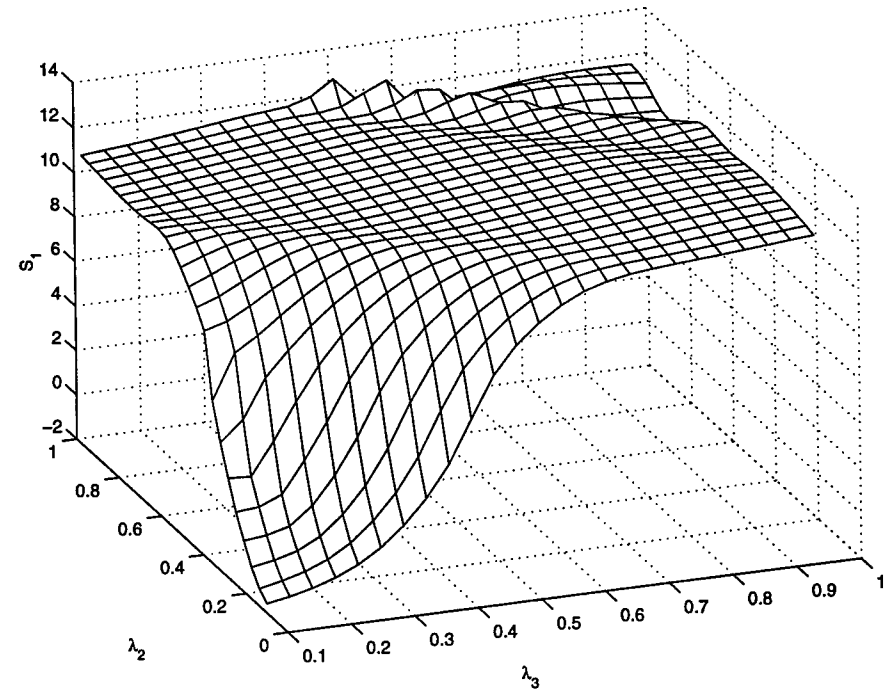


Figure 30: CCNN(I): Output S_1 vs. (λ_2, λ_3), $\lambda_1 = 0.3, \theta = 10$

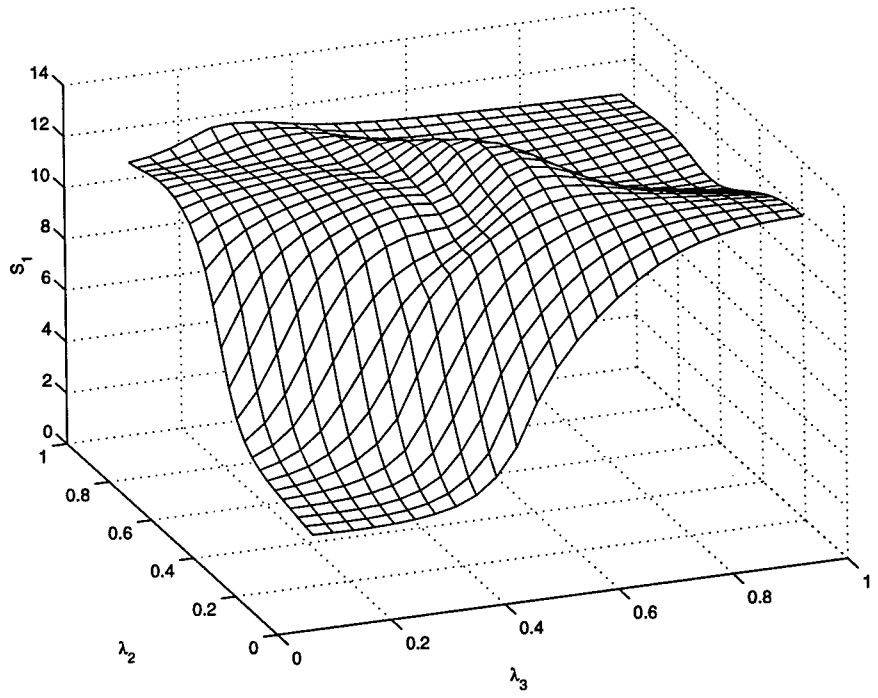


Figure 31: CCNN(II): Output S_1 vs. (λ_2, λ_3), $\lambda_1 = 0.3, \theta = 10$

	49 point set	training set (size:500)	test set (size:500)
Polynomial	102.0 (80.0%)	8.073 (21.9%)	10.83 (22.0%)
	49 point set	training set (size:1000)	test set (size:500)
CCNN(II)	80.26 (23.2%)	7.929 (12.7%)	11.35 (13.7%)

Table 3: CCNN and polynomial trained using larger data set

5 CONCLUSIONS AND FUTURE RESEARCH DIRECTIONS

In this section, we summarize the main findings, lessons learned, and recommendations for future directions that have resulted from this project.

Concurrent and Parallel Simulation. A basic theoretical framework underlying concurrent simulation algorithms is one of the accomplishments of this project, capitalizing and extending the results of our earlier work. Based on this framework, we have developed a general (i.e., *not limited* by specific modeling assumptions) concurrent simulation approach implemented through the *Time Warping Algorithm* (TWA) and quantified and compared the effectiveness of this algorithm to conventional repetitive simulation through the concept of "speedup". We have found that the speedups achieved by concurrent simulation are significantly affected by such factors as the particular computer hardware used and the data structures selected. This suggests the need for a further exploration of different ways to implement the TWA and its variants.

Application of concurrent simulation techniques to parallel processing environments promises at least one additional order of magnitude in speedup over concurrent simulation on sequential processors. The integration of computer simulation methodologies with parallel processing architectures is a direction that holds great promise.

Stochastic Fidelity and Multi-resolution Simulation. One of the accomplishments of this project is to document and substantiate the *stochastic fidelity* issue that arises in hierarchical simulation. Simple averaging of the output data from a high resolution simulator to generate input data for a low resolution simulator is inadequate and occasionally dramatically erroneous. Resolving this issue is associated with the problem of systematic clustering methods that "bundle" high resolution output data in a way that incorporates the statistical

information lost through simple averaging. Our research over various clustering methods has led us to the path bundle grouping approach based on Adaptive Resonance Theory presented in Section 3.2.4. A crucial research direction to be pursued in the future is the development of fast clustering algorithms to group the huge dimensional data generated from typical high resolution simulators. Our investigation of the ART2 neural network for this purpose has led to promising results and identified key issues to be addressed.

The nature of this line of research is such that significant progress is made through extensive empirical work and experimentation with benchmark problems. We therefore believe that such problems need to be defined and thoroughly explored at the same time as specific analytical or numerical tools are developed.

Metamodeling through Neural Networks. The metamodeling procedure we have studied combines simulation of a complex system with the process of training a neural network to become a surrogate model of this system. This exploits the ability of a neural network to act as a universal function approximator. Using a Cascade Correlation Neural Network (CCNN), a multilayer neural network that builds itself while it learns, we were able to study concrete models and problems demonstrating that neural network metamodels are significantly more accurate than their polynomial counterparts, especially when asymptotic behavior in input-output relationships is involved; such is the case in the ARMS case, as detailed in Sections 4.2 and 4.3.

Important issues we have identified in this component of our project include: (a) Determining the training data size for metamodeling through neural networks. An interesting direction, for example, is the investigation of adaptive mechanisms and segmentation of a problem to develop separate models for different regions of the input space. (b) Improving the learning efficiency and speeding up the learning process of a neural network. As an example, starting from several different initial points (multi-starts) seems to substantially

help training, but has yet to be studied in a systematic framework.

References

[1] J.S. Ahn, P.B. Danzig, "Packet Network Simulation: Speedup and Accuracy Versus Timing Granularity," *IEEE/ACM Trans. on Networking*, **4**, 5, 743–757, Oct. 1996.

[2] M. Blatt, S. Wiseman, and E. Domany, "Superparamagnetic clustering of data", Vol. 76, *Phys. Rev. Lett.*, p. 3251, 1996.

[3] Carpenter, G.A. and Stephen Grossberg, "ART 2: self-organization of stable category recognition codes for analog input patterns", Applied Optics, December 1987.

[4] Cassandras, C.G.,"Discrete Event Systems, Modeling and Performance Analysis", *Irwin Publ.*, (1993).

[5] C.G. Cassandras, and W.-B. Gong, "Enabling Technologies for Real-Time Simulation", Technical Report, Dec. 1996

[6] C.G. Cassandras, W.-B. Gong, C. Liu, C. Panayiotou, D. Pepyne, "Simulation-driven metamodeling of complex systems using neural networks", *Proceedings of 19th SPIE Conference*, April, 1998.

[7] C.G.Cassandras and C.G.Panayiotou, C.G., "Concurrent Sample Path Analysis of Discrete Event Systems", J. of Discrete Event Dynamic Systems, Vol. 9, 2, pp. 171-195, 1999.

[8] D. Caughlin, "Verification, Validation, and Accreditation of Models and Simulations through Reduced Order Metamodels," in Proceedings of the 1995 Winter Simulation Conference, pp. 1405-1412, December 1995.

[9] L. Fausett, Fundamentals of Neural Networks, Architecture, Algorithms and Applications, Prentice Hall, 1994.

[10] Fahlman, S.E., and Lebiere, C., "The Cascade-Correlation Learning Architecture", Carnegie Mellon Univ., Technical Report CMU-CS-90-100, Feb., 1990.

[11] Floyd, S., and Jacobson, V., "Random Early Detection gateways for Congestion Avoidance", *IEEE/ACM Trans. on Networking*, **1**, 397–413, Aug. 1993.

[12] M.I. Freidlin and A.D. Wentzell, *Random Perturbations of Dynamical Systems*, Springer-Verlag, 1984.

[13] Gear,C.W.,"The Numerical Integration of Ordinary Differencial Equations," *Mathematics of Computation*, Vol.21, No.98, April 1967, pp.146-156.

[14] P. Glynn, Private Communications.

[15] M. Hoehfeld and S.E. Fahlman, "Learning with Limited Numerical Precision Using the Cascade-Correlation Algorithm", IEEE Trans. on Neural Networks, Vol. 3., No. 4, July, 1992.

[16] K. Gokbayrak and C. G. Cassandras, "Stochastic Discrete Optimization Using a Surrogate Problem Methodology", *Proceedings of 20th SPIE Conference*, Orlando, April, 1999.

[17] Y. Guo, X. Yin and W. Gong, "ART2 Neural Network Clustering for Hierarchical Simulation", *Proceedings of 19th SPIE Conference*, Orlando, April, 1998.

[18] Y. Kifer, *Random Perturbations of Dynamical Systems*, Birkhauser, 1988.

[19] S.C. Lee and T.W. Tang, "Transport Coefficients for a Silicon Hydrodynamic Model Extracted from Inhomogeneous Monte Carlo Calculations", Solid-State Electronics, Vol. 35, No.4, pp.561-569, 1992.

[20] Phatak, D.S., Koren, I., "Connectivity and performance Tradeoffs in the Cascade Correlation Learning Architecture", Univ. of Mass, Technical Report No. TR-92-CSE-27, Aug., 1992.

[21] J. S. Prsemienicki, *Mathematical Methods in Defence Analyses*, 2nd Ed., AIAA Education Series, 1994.

[22] F.A. Tatum, "A Tactical Electronic Reconnaissance Simulation Model," Rand Corporation Technical Report, 24 October 1969

[23] U.S. Army Concept Analysis Agency, "Concept Evaluation Model", 1983.

[24] T. L. Vincent, and W. J. Grantham, *Nonlinear and Optimal Control Systems*, John Wiley & Sons, Inc., 1997.

[25] A. Yan and W. Gong, "Fluid Simulation of High Speed Networks", To appear, *IEEE Transactions on Information Theory*, June, 1999.

[26] M. A. Zeimer, and J.D. Tew, "Metamodel Applications Using TERSM," in Proceedings of the 1995 Winter Simulation Conference, pp. 1421-1428, December 1995.

[27] M.A. Zeimer, J.D. Tew, R.G. Sargent, and A. Sisti, "Metamodel Procedures for Air Engagement Simulation Models," IRAE Technical Report, Rome Laboratory, Griffis A.F.B., N.Y., January 1993.

APPENDIX

This Appendix contains reprints of publications that are relevant and provide supporting documentation for this report.

Discrete Event Dynamic Systems: Theory and Applications, 9, 171–195 (1999)

Concurrent Sample Path Analysis of Discrete Event Systems

CHRISTOS G. CASSANDRAS cgc@enga.bu.edu
Department of Manufacturing Engineering, Boston University, Boston, MA 02215

CHRISTOS G. PANAYIOTOU panayiot@ecs.umass.edu
Department of Electrical and Computer Engineering, University of Massachusetts, Amherst, MA 01003

Abstract. The sample path constructability problem for Discrete Event Systems (DES) involves the observation of a sample path under a particular parameter value θ of the system with the requirement to concurrently construct multiple sample paths of the DES under different values *using only information available along the given sample path*. This allows the on-line estimation of performance measures $J(\theta)$, not available in closed form, over a range of values of θ. We present a sample path coupling approach that solves the problem without imposing any restrictions on the event processes in the system. A specific "time warping" algorithm is described and its performance is analyzed in terms of computational cost. Our approach is illustrated through a number of simulation results.

Keywords: discrete event systems, sample path construction, concurrent estimation, timed state automaton

1. Introduction

A typical problem one faces in the design, control, and optimization of Discrete Event Systems (DES) is that of determining how some performance measure $J(\theta)$ varies as a function of some parameter θ. As a rule, analytical expressions for $J(\theta)$ are simply unavailable, thus forcing one to resort to repetitive simulation or on-line trial-and-error techniques. This requires the generation of a sample path of the system at least once for every value of θ of interest. It is easy to see how tedious (if at all feasible) this process becomes, especially when θ is a vector and the DES is stochastic, requiring large number of sample paths to achieve desired levels of statistical accuracy.

It is by now well-documented in the literature that the nature of sample paths of DES can be exploited so as to extract a significant amount of information, beyond merely an estimate of $J(\theta)$. It has been shown that observing a sample path under some parameter value θ allows us to efficiently obtain estimates of derivatives of the form $dJ/d\theta$ which are in many cases unbiased and strongly consistent (e.g., see Cassandras, 1993; Glasserman, 1991; Ho and Cao, 1991) where Infinitesimal Perturbation Analysis (IPA) and its extensions are described). Similarly, Finite Perturbation Analysis (FPA) has been used to estimate finite differences of the form $\Delta J(\Delta\theta)$ or to approximate the derivative $dJ/d\theta$ through $\Delta J/\Delta\theta$ when other PA techniques fail. Of particular interest are often parameters θ that take values from a discrete set $\{\theta_1, \ldots, \theta_m\}$ (e.g., queueing capacities, threshold values in certain control policies), in which case we desire to effectively construct sample paths under any $\theta_1, \ldots, \theta_m$ by just observing a sample path under one of these parameter values. All of the methods developed to date, regardless of specific details, have been motivated by the same objective:

From a single sample path under θ extract information to estimate the derivative $dJ/d\theta$ or the response of the system, $J(\theta')$, under other parameter values $\theta' \neq \theta$. This information can be extremely useful in sensitivity analysis and optimization of DES. It is also the case that structural properties of a DES, such as monotonicity and convexity, are revealed by this type of sample path analysis (see also Glasserman and Yao, 1994).

In this paper, we will concentrate on the general *sample path constructability* problem for DES. That is, given a sample path under a particular parameter value θ, the problem is to construct multiple sample paths of the system under different values *using only information available along the given sample path.* A solution to this problem can be obtained when the system under consideration satisfies the *Constructability Condition (CO)* presented in Cassandras and Strickland (1989a,b). Suppose that a sample path of the system is observed under parameter θ and we would like to construct the corresponding sample path under some θ'. Then *(CO)* consists of two parts. The first part is the *Observabilty Condition (OB)* which states that at every state the feasible event set of the constructed sample path must be a subset of the feasible event set of the observed sample path. The second part is a requirement that all lifetimes of feasible events conditioned on event ages are equal in distribution.

Unfortunately, *(CO)* is not easily satisfied. Nonetheless, two methods to date have been developed making it possible to construct multiple sample paths at different parameter settings from a single sample path at some extra cost. In particular, the *Standard Clock (SC)* approach (Vakili, 1991) solves the sample path constructability problem for models with exponentially distributed event lifetimes by exploiting the well-known uniformization technique for Markov chains. This approach allows the *concurrent* construction of multiple sample paths under different (continuous or discrete) parameters at the expense of introducing "fictitious" events. Chen and Ho (1995) have proposed a *generalized SC* approach that uses approximation techniques to extend the *SC* approach to systems with non-exponential event lifetime distributions. On the other hand, *Augmented System Analysis (ASA)* (Cassandras and Strickland, 1989a,b) solves the constructability problem by "suspending" the construction of one or more paths during certain segments of the observed sample path in a way such that the stochastic characteristics of the observed sample path are preserved. In *ASA*, it is still necessary to assume exponential event lifetime distributions, although, with a minor extension it is possible to allow at most one event to have a non-exponential lifetime distribution (see Cassandras, 1993; Cassandras and Strickland, 1989b for details).

In this paper we develop and analyze an approach for solving the constructability problem for general DES (Cassandras and Panayiotou, 1996). We emphasize that the proposed scheme is suited to *on-line* sample path construction, where actual system data are processed for performance estimation purposes. In contrast, the *SC* approach is only relevant in the context of simulation-based analysis (although it is possible to adapt the *SC* to on-line applications (Cassandras et al., 1990), but with considerable effort). Moreover, unlike *SC* and *ASA*, our approach can be used in systems with arbitrary lifetime distributions without involving any of the approximations suggested in the *generalized SC* method in (Chen and Ho, 1995). The central idea of our approach is quite simple: when an event on the observed sample path under θ occurs and causes a state transition, its lifetime is stored. These stored lifetimes are used as the input to multiple concurrent sample path

generators (e.g., simulators) under different parameter values $\theta' \neq \theta$. Thus, instead of separately generating lifetimes for each event which is common in the observed and each constructed sample path we use the lifetimes that have been directly observed.[1] Whenever a constructed sample path enters a state such that the feasible event set includes events which have not yet occurred on the observed sample path (i.e., no lifetimes have been observed for these events), the construction is suspended until the required events and corresponding lifetimes become available on the nominal path. Therefore, the constructability condition is bypassed at the expense of a process for starting/stopping each sample path construction. In this paper we will define this process, study its properties and compare its effectiveness to other approaches. The viewpoint we adopt in the sample path constructability problem is one of coupling the observed process under θ to all processes under different $\theta' \neq \theta$ and then deriving event-driven dynamics that describe this coupling. Also, note that in certain cases event lifetimes may not be directly observable, in which case they need to be recovered from the output available, a problem referred to as *invertibility* (Park and Chong, 1995).

It is worth pointing out that this *concurrent sample path constructability* approach can be used in two different modes. Primarily, it can be used on-line as a *concurrent estimation* scheme. In addition, for a class of systems, it can be used off-line as a *concurrent simulation* approach. In the case of *concurrent estimation*, the scheme observes a real DES under some parameter value θ and estimates the system's performance under a set of hypothetical parameter values $\theta_1, \ldots, \theta_M$. These estimates can be used together with optimization schemes requiring such information (e.g., Yan and Mukai, 1992; Gong et al., 1992; Cassandras and Julka, 1994, Panayiotou and Cassandras, 1996); also, see (Panayiotou and Cassandras, 1997) for an application of the scheme in a dynamic resource allocation problem). In the case of *concurrent simulation*, the algorithm developed is incorporated into a simulation environment in order to generate performance estimates under a range of parameters *faster* than repeatedly simulating the system under each parameter (what is commonly referred to as "brute-force simulation").

The paper is organized as follows. In section 2 we formally define the general sample path constructability problem. In section 3 we describe the process through which the problem is solved by coupling an observed sample path to multiple concurrently generated sample paths under different parameter settings, and a detailed procedure, the *Time Warping Algorithm* is presented. An evaluation of the proposed algorithm is presented in section 4. Some extensions of the algorithm are discussed in section 5 and several simulation results are included in section 6. Finally we close with conclusions from this work in section 7.

2. Problem Definition

We consider a DES and adopt the modeling framework of a *stochastic timed state automaton* $(\mathcal{E},\mathcal{X},\Gamma,f,x_0)$ (Cassandras, 1993). Here, $\mathcal{E}$ is a countable event set, $\mathcal{X}$ is a countable state space, and $\Gamma(x)$ is a set of feasible (or enabled) events, defined for all $x \in \mathcal{X}$ such that $\Gamma(x) \subseteq \mathcal{E}$. The state transition function $f(x, e)$ is defined for all $x \in \mathcal{X}$, $e \in \Gamma(x)$, and specifies the next state resulting when e occurs at state x. Finally, x_0 is a given initial state. In addition, for simplicity we assume that the DES satisfies the non-interruption condition,

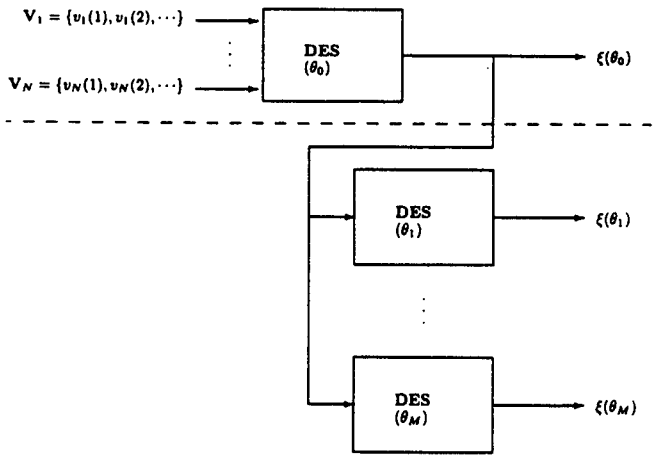


Figure 1. The sample path constructability problem for DES.

i.e., once an event is enabled it cannot be disabled; this is not essential to the derivation of our results however.

Remark. The definition is easily modified to $(\mathcal{E},\mathcal{X},\Gamma,p,p_0)$ in order to include probabilistic state transition mechanisms. In this case, the state transition probability $p(x';x,e')$ is defined for all $x, x' \in \mathcal{X}$, $e' \in \mathcal{E}$, and is such that $p(x';x,e') = 0$ for all $e' \notin \Gamma(x)$. In addition, $p_0(x)$ is the pmf $P[x_0 = x]$, $x \in \mathcal{X}$, of the initial state x_0.

Assuming the cardinality of the event set $\mathcal{E}$ is N, the input to the system is a set of event lifetime sequences $\{\mathbf{V}_1, \ldots, \mathbf{V}_N\}$, one for each event, where $\mathbf{V}_i = \{v_i(1), v_i(2), \ldots\}$ is characterized by some arbitrary distribution. Under some system parameter θ_0, the output is a sequence $\xi(\theta_0) = \{(e_k, t_k),\ k = 1, 2, \ldots\}$ where $e_k \in \mathcal{E}$ is the kth event and t_k is its corresponding occurrence time (see Figure 1). Based on any observed $\xi(\theta_0)$, we can evaluate $L[\xi(\theta_0)]$, a sample performance metric for the system. For a large family of performance metrics of the form $J(\theta_0) = E[L[\xi(\theta_0)]]$, $L[\xi(\theta_0)]$ is therefore an estimate of $J(\theta_0)$. Defining a set of parameter values of interest $\{\theta_0, \theta_1, \ldots, \theta_M\}$, the sample path constructability problem is:

> *For a DES under θ_0, construct all sample paths $\xi(\theta_1), \ldots, \xi(\theta_M)$ given a realization of lifetime sequences $\mathbf{V}_1, \ldots, \mathbf{V}_N$ and the sample path $\xi(\theta_0)$.*

For simplicity, in the rest of this paper we assume that the DES under investigation satisfies the following three assumptions. Extensions allowing the relaxation of these assumptions are possible and are briefly described in section 5.

- *(A1) Feasibility Assumption*: Let x_n be the state of the DES after the occurrence of the nth event. Then, for any n, there exists at least one $r > n$ such that $e \in \Gamma(x_r)$ for any $e \in \mathcal{E}$.

- (*A2*) *Invariability Assumption*: Let $\mathcal{E}$ be the event set under the nominal parameter θ_0 and let $\mathcal{E}_m$ be the event set under $\theta_m \neq \theta_0$. Then, $\mathcal{E}_m = \mathcal{E}$.
- (*A3*) *Similarity Assumption*: Let $G_i(\theta_0), i \in \mathcal{E}$ be the event lifetime distribution for the event i under θ_0 and let $G_i(\theta_m), i \in \mathcal{E}$ be the corresponding event lifetime distribution under θ_m. Then, $G_i(\theta_0) = G_i(\theta_m)$ for all $i \in \mathcal{E}$.

Assumption *A1* guarantees that in the evolution of any sample path all events in $\mathcal{E}$ will always become feasible at some point in the future. If for some DES assumption *A1* is not satisfied, i.e. there exists an event α that never gets activated after some point in time, then, as we will see, it is possible that the construction of some sample path will remain suspended forever waiting for α to happen. Note that a DES with an irreducible state space immediately satisfies this condition.

Assumption *A2* states that changing a parameter from θ_0 to some $\theta_m \neq \theta_0$ does not alter the event set $\mathcal{E}$. More importantly, *A2* guarantees that changing to θ_m does not introduce any new events so that all event lifetimes for all events can be observed from the nominal sample path.

Finally, assumption *A3* guarantees that changing a parameter from θ_0 to some $\theta_m \neq \theta_0$ does not affect the distribution of one or more event lifetime sequences. This allows us to use exactly the same lifetimes that we observe in the nominal sample path to construct the perturbed sample path. In other words, our analysis focuses on *structural* system parameters rather that *distributional* parameters. As we will see, however, it is straightforward to handle the latter at the expense of some computational cost.

3. Coupled Sample Path Construction

Before presenting the coupling approach we use to solve the constructability problem and the explicit procedure we will refer to as the *Time Warping Algorithm*, let us consider the following motivating example in order to illustrate the main aspects of this procedure.

In a $G/G/1/K$ queueing system, the event set is $\mathcal{E} = \{a, d\}$ (a for arrival, d for departure), and the state x is a non-negative integer with $x \leq K$ representing the number of customers currently in the system. Let the observed sample path be one with queue capacity $K = 2$, and let us try to construct a sample path under $K = 3$ in the framework of Figure 1. Let $\Gamma(x[K])$ be the feasible event set at state x for the system under K and assume that both systems are initially empty. Unlike earlier sample path constructability techniques, i.e., the *SC* and *ASA* methods, we can no longer maintain between the two sample paths (the observed one and the one to be constructed) a coupling which preserves full synchronization of events; this is because of the absence of Markovian event processes which allow us to exploit the memoryless property. Thus, we must maintain each feasible event set, $\Gamma(x[2])$ and $\Gamma(x[3])$, separately for each observed state $x[2]$ and constructed state $x[3]$. Whenever an event is observed, its lifetime is assumed to become available (i.e., the time when this event was activated is known). Each such lifetime is subsequently used in the construction of the sample path under $K = 3$.

To see precisely how this can be done, we start out with a state $x[3] = 0$ for the *constructed* sample path, so that $\Gamma(x[3]) = \{a\}$. Since no event lifetimes are initially available, we

consider the sample path of this system as "suspended". The initial state of the *observed* sample path is $x[2] = 0$ so that $\Gamma(x[2]) = \{a\}$. Therefore, the first observed event is a. At this point, the constructed sample path may be "resumed", since all lifetimes of the events in $\Gamma(x[3])$ are now available, namely the lifetime of a. The constructed sample path advances time, and updates its state to $x[3] = 1$. Now $\Gamma(x[3]) = \{a, d\}$ but neither event has been observed yet, so the constructed sample path is suspended again until at least one a *and* one d events occur at the observed sample path. This start/stop (or suspend/resume) process goes on until a sample path under $K = 3$ is constructed up to a desired number of events or some specified time. Note that the coupling between the two sample paths requires a process through which, at every observed event, one must determine whether it is possible to resume construction of the suspended path. It may take several observed events before this is possible. However, it is also possible that such an event triggers a series of events on the constructed sample path, and hence a sequence of state transitions and time updates (e.g., if $\Gamma(x[3]) = \{a, d\}$, a sequence of events $\{a, a, a, d\}$ will cause three state transitions in a row as soon as d is observed). The fact that in this process we move backward in time to revisit a suspended sample path and then forward by one or more event occurrences lends itself to the term "time warping".

Despite the simplicity of the main concept, the formal description of the coupling scheme and the process through which the start/stop condition is determined are somewhat tedious largely due to the notation necessary. In what follows, we introduce some basic definitions and notation before deriving the coupling process dynamics and describing the exact sample path construction procedure.

3.1. Notation and Definitions

First, let $\xi(n, \theta) = \{e_j: j = 1, \ldots, n\}$, with $e_j \in \mathcal{E}$, be the sequence of events that constitute the observed sample path up to n total events. Although $\xi(n, \theta)$ is clearly a function of the parameter θ, we will write $\xi(n)$ to refer to the observed sample path and adopt the notation $\hat{\xi}(k) = \{\hat{e}_j: j = 1, \ldots, k\}$ for any constructed sample path under a different value of the parameter up to k events in that path. It is important to realize that k is actually a function of n, since the constructed sample path is coupled to the observed sample path through the observed event lifetimes; however, again for the sake of notational simplicity, we will refrain from continuously indicating this dependence.

Next we define the *score* of an event $i \in \mathcal{E}$ in a sequence $\xi(n)$, denoted by $s_i^n = [\xi(n)]_i$, to be the non-negative integer that counts the number of instances of event i in this sequence. The corresponding score of i in a constructed sample path is denoted by $\hat{s}_i^k = [\hat{\xi}(k)]_i$. In what follows, all quantities with the symbol " $\hat{\cdot}$ " refer to a typical constructed sample path.

Associated with every event type $i \in \mathcal{E}$ in $\xi(n)$ is a sequence of s_i^n event lifetimes

$$\mathbf{V}_i(n) = \{v_i(1), \ldots, v_i(s_i^n)\} \quad \text{for all } i \in \mathcal{E}$$

The corresponding set of sequences in the constructed sample path is:

$$\hat{\mathbf{V}}_i(k) = \{v_i(1), \ldots, v_i(\hat{s}_i^k)\} \quad \text{for all } i \in \mathcal{E}$$

which is a subsequence of $\mathbf{V}_i(n)$ with $k \leq n$. In addition, we define the following sequence of lifetimes:

$$\tilde{\mathbf{V}}_i(n,k) = \{v_i(\hat{s}_i^k + 1), \ldots, v_i(s_i^n)\} \quad \text{for all } i \in \mathcal{E}$$

which consists of all event lifetimes that are in $\mathbf{V}_i(n)$ but not in $\hat{\mathbf{V}}_i(k)$. Associated with any one of these sequences are the following operations. Given some $\mathbf{W}_i = \{w_i(j), \ldots, w_i(r)\}$,

Suffix Addition: $\mathbf{W}_i + \{w_i(r+1)\} = \{w_i(j), \ldots, w_i(r), w_i(r+1)\}$ and,

Prefix Subtraction: $\mathbf{W}_i - \{w_i(j)\} = \{w_i(j+1), \ldots, w_i(r)\}$.

Note that the addition and subtraction operations are defined so that a new element is always added as the *last* element (the *suffix*) of a sequence, whereas subtraction always removes the *first* element (the *prefix*) of the sequence.

Next, define the set

$$A(n,k) = \left\{i:\ i \in \mathcal{E}, s_i^n > \hat{s}_i^k\right\} \tag{1}$$

which is associated with $\tilde{\mathbf{V}}_i(n,k)$ and consists of all events i whose corresponding sequence $\tilde{\mathbf{V}}_i(n,k)$ contains at least one element. Thus, every $i \in A(n,k)$ is an event that has been observed in $\xi(n)$ and has at least one lifetime that has yet to be used in the coupled sample path $\hat{\xi}(k)$. Hence, $A(n,k)$ should be thought of as the set of *available* events to be used in the construction of the coupled path.

Finally, we define the following set, which is crucial in our approach:

$$M(n,k) = \Gamma(\hat{x}_k) - \left(\Gamma(\hat{x}_{k-1}) - \{\hat{e}_k\}\right) \tag{2}$$

where, clearly, $M(n,k) \subseteq \mathcal{E}$. Note that $\hat{e}_k$ is the triggering event at the $(k-1)$th state visited in the constructed sample path. Thus, $M(n,k)$ contains all the events that are in the feasible event set $\Gamma(\hat{x}_k)$ but not in $\Gamma(\hat{x}_{k-1})$; in addition, $\hat{e}_k$ also belongs to $M(n,k)$ if it happens that $\hat{e}_k \in \Gamma(\hat{x}_k)$. Intuitively, $M(n,k)$ consists of all *missing* events from the perspective of the constructed sample path when it enters a new state $\hat{x}_k$: those events already in $\Gamma(\hat{x}_{k-1})$ which were not the triggering event remain available to be used in the sample path construction as long as they are still feasible; all other events in the set are "missing" as far as residual lifetime information is concerned.

The concurrent sample path construction process we are interested in consists of two coupled processes, each generated by a timed state automaton. This implies that there are two similar sets of equations that describe the dynamics of each process. In addition, we need a set of equations that captures the coupling between them.

3.2. *Timed State Automaton Dynamics*

We briefly review here the standard timed state automaton dynamics, also known as a Generalized Semi-Markov Scheme (GSMS) (see Cassandras, 1993; Glasserman, 1991; Ho and Cao, 1991). We introduce two additional variables, t_n to be the time when the nth event

occurs, and $y_i(n)$, $i \in \Gamma(x_n)$, to be the residual lifetime of event i after the occurrence of the nth event (i.e., it is the time left until event i occurs). On a particular sample path, just after the nth event occurs the following information is known: the state x_n from which we can determine $\Gamma(x_n)$, the time t_n, the residual lifetimes $y_i(n)$ for all $i \in \Gamma(x_n)$, and all event scores s_i^n, $i \in \mathcal{E}$. The following equations describe the dynamics of the timed state automaton.

Step 1: Determine the smallest residual lifetime among all feasible events at state x_n, denoted by y_n^*:

$$y_n^* = \min_{i \in \Gamma(x_n)} \{y_i(n)\} \tag{3}$$

Step 2: Determine the triggering event:

$$e_{n+1} = \arg \min_{i \in \Gamma(x_n)} \{y_i(n)\} \tag{4}$$

Step 3: Determine the next state:

$$x_{n+1} = f(x_n, e_{n+1}) \tag{5}$$

Step 4: Determine the next event time:

$$t_{n+1} = t_n + y_n^* \tag{6}$$

Step 5: Determine the new residual lifetimes for all new feasible events $i \in \Gamma(x_{n+1})$:

$$y_i(n+1) = \begin{cases} y_i(n) - y_n^* & \text{if } i \neq e_{n+1} \text{ and } i \in \Gamma(x_n) \\ v_i(s_i^n + 1) & \text{if } i = e_{n+1} \text{ or } i \notin \Gamma(x_n) \end{cases} \quad \text{for all } i \in \Gamma(x_{n+1}) \tag{7}$$

Step 6: Update the event scores:

$$s_i^{n+1} = \begin{cases} s_i^n + 1 & \text{if } i = e_{n+1} \\ s_i^n & \text{otherwise} \end{cases} \tag{8}$$

Equations (3)–(8) describe the sample path evolution of a timed state automaton. These equations apply to both the observed and the constructed sample paths. Next, we need to specify the mechanism through which these two sample paths are coupled in a way that enables event lifetimes from the observed $\xi(n)$ to be used to construct a sample path $\hat{\xi}(k)$. First, observe that the process described by (3)–(8), applied to $\hat{\xi}(k)$, hinges on the availability of residual lifetimes $\hat{y}_i(k)$ for all $i \in \Gamma(\hat{x}_k)$. Thus, the constructed sample path can only be "active" at state $\hat{x}_k$ if every $i \in \Gamma(\hat{x}_k)$ is such that either $i \in (\Gamma(\hat{x}_{k-1}) - \{\hat{e}_k\})$ (in which case $\hat{y}_i(k)$ is a residual lifetime of an event available from the previous state transition) or $i \in A(n, k)$ (in which case a full lifetime of i is available from the observed sample path). This motivates the following:

Definition 1. A constructed sample path is *active* at state $\hat{x}_k$ after the occurrence of an observed event e_n if, for every $i \in \Gamma(\hat{x}_k)$, $i \in (\Gamma(\hat{x}_{k-1}) - \{\hat{e}_k\}) \cup A(n,k)$.

Thus, the start/stop conditions for the construction of a sample path are determined by whether it is active at the current state or not.

3.3. *Coupling Dynamics*

Upon occurrence of the $(n+1)$th observed event, e_{n+1}, the first step is to update the event lifetime sequences $\tilde{\mathbf{V}}_i(n,k)$ as follows:

$$\tilde{\mathbf{V}}_i(n+1,k) = \begin{cases} \tilde{\mathbf{V}}_i(n,k) + v_i(s_i^n) & \text{if } i = e_{n+1} \\ \tilde{\mathbf{V}}_i(n,k) & \text{otherwise} \end{cases} \tag{9}$$

The addition of a new event lifetime implies that the "available event set" $A(n,k)$ defined in (1) may be affected. Therefore, it is updated as follows:

$$A(n+1,k) = A(n,k) \cup \{e_{n+1}\} \tag{10}$$

Finally, note that the "missing event set" $M(n,k)$ defined in (2) remains unaffected by the occurrence of observed events:

$$M(n+1,k) = M(n,k) \tag{11}$$

At this point, we are able to decide whether all lifetime information to proceed with a state transition in the constructed sample path is available or not. In particular, the condition

$$M(n+1,k) \subseteq A(n+1,k) \tag{12}$$

may be used to determine whether the constructed sample path is active at the current state $\hat{x}_k$ (in the sense of Definition 1). The following is a formal statement of this fact.

LEMMA 1 *A constructed sample path is active at state $\hat{x}_k$ after an observed event e_{n+1} if and only if $M(n+1,k) \subseteq A(n+1,k)$.*

Proof: Let $B_k = \Gamma(\hat{x}_{k-1}) - \{\hat{e}_k\}$. Suppose $M(n+1,k) \subseteq A(n+1,k)$ and let $i \in \Gamma(\hat{x}_k)$. Then, consider the following two cases: *(i)* If $i \in B_k$, then, by definition, the sample path is active at $\hat{x}_k$. *(ii)* If $i \notin B_k$, by the definition of $M(n,k)$ in (2), it follows that $i \in M(n,k) = M(n+1,k)$ and since $M(n+1,k) \subseteq A(n+1,k)$, we have $i \in A(n+1,k)$ which implies again that the sample path is active at state $\hat{x}_k$.

Conversely, suppose that the sample path is active at state $\hat{x}_k$. Let $i \in M(n+1,k)$. Then, from (2), $i \in \Gamma(\hat{x}_k)$, but $i \notin B_k$. However, since the sample path is active, we must have $i \in A(n+1,k)$ by Definition 1. Therefore, $M(n+1,k) \subseteq A(n+1,k)$. ■

Assuming (12) is satisfied, equations (3)–(8) may be used to update the state $\hat{x}_k$ of the constructed sample path. In so doing, lifetimes $v_i(s_i^k + 1)$ for all $i \in M(n+1,k)$ are used

from the corresponding sequences $\tilde{\mathbf{V}}_i(n+1,k)$. Thus, upon completion of the six state update steps, all three variables associated with the coupling process, i.e., $\tilde{\mathbf{V}}_i(n,k)$, $A(n,k)$, and $M(n,k)$ need to be updated. In particular,

$$\tilde{\mathbf{V}}_i(n+1,k+1) = \begin{cases} \tilde{\mathbf{V}}_i(n+1,k) - v_i(\hat{s}_i^k + 1) & \text{for all } i \in M(n+1,k) \\ \tilde{\mathbf{V}}_i(n+1,k) & \text{otherwise} \end{cases} \tag{13}$$

This operation immediately affects the set $A(n+1,k)$ which is updated as follows:

$$A(n+1,k+1) = A(n+1,k) - \left\{i:\ i \in M(n+1,k),\ \hat{s}_i^{k+1} = s_i^{n+1}\right\} \tag{14}$$

Finally, applying (2) to the new state $\hat{x}_{k+1}$,

$$M(n+1,k+1) = \Gamma(\hat{x}_{k+1}) - \left(\Gamma(\hat{x}_k) - \{\hat{e}_{k+1}\}\right) \tag{15}$$

Therefore, we are again in a position to check condition (12) for the new sets $M(n+1,k+1)$ and $A(n+1,k+1)$. If it is satisfied, then we can proceed with one more state update on the constructed sample path; otherwise, we wait for the next event on the observed sample path until (12) is again satisfied. Similar to Lemma 1, we have:

LEMMA 2 *A constructed sample path is active at state $\hat{x}_{k+1}$ after event $\hat{e}_{k+1}$ if and only if $M(n+1,k+1) \subseteq A(n+1,k+1)$.*

Proof: This is similar to the proof of Lemma 1. Let $B_{k+1} = \Gamma(\hat{x}_k) - \{\hat{e}_{k+1}\}$. Suppose $M(n+1,k+1) \subseteq A(n+1,k+1)$ and let $i \in \Gamma(\hat{x}_{k+1})$. Then, consider the following two cases: *(i)* If $i \in B_{k+1}$, then, by definition, the sample path is active at state $\hat{x}_{k+1}$. *(ii)* If $i \notin B_{k+1}$, by (15), it follows that $i \in M(n+1,k+1)$ and since $M(n+1,k+1) \subseteq A(n+1,k+1)$, we have $i \in A(n+1,k+1)$ which implies again that the sample path is active at state $\hat{x}_{k+1}$.

Conversely, suppose that $\hat{x}_{k+1}$ is active. Let $i \in M(n+1,k+1)$. Then, from (15), $i \in \Gamma(\hat{x}_{k+1})$, but $i \notin B_{k+1}$. However, since the sample path is active, we must have $i \in A(n+1,k+1)$ by Definition 1. Therefore, $M(n+1,k+1) \subseteq A(n+1,k+1)$. ■

The analysis above is summarized below in the form of the following *Time Warping Algorithm (TWA).*

Time Warping Algorithm (*TWA*):

1. INITIALIZE

$n := 0, k := 0, t_n := 0, \hat{t}_k := 0, x_n := x_0, \hat{x}_k = \hat{x}_0,$
$y_i(n) = v_i(1)$ for all $i \in \Gamma(x_n)$, $s_i^n = 0, \hat{s}_i^k = 0$ for all $i \in \mathcal{E}$,
$M(0,0) := \Gamma(\hat{x}_0)$, $A(0,0) := \emptyset$

2. WHEN EVENT e_n IS OBSERVED:

2.1 Use (3)–(8) to determine $e_{n+1}, x_{n+1}, t_{n+1}, y_i(n+1)$ for all $i \in \Gamma(x_{n+1})$, s_i^{n+1} for all $i \in \mathcal{E}$.

2.2 Add the e_{n+1} event lifetime to $\tilde{\mathbf{V}}_i(n+1,k)$:

$$\tilde{\mathbf{V}}_i(n+1,k) = \begin{cases} \tilde{\mathbf{V}}_i(n,k) + v_i(s_i^n) & \text{if } i = e_{n+1} \\ \tilde{\mathbf{V}}_i(n,k) & \text{otherwise} \end{cases}$$

2.3 Update the available event set $A(n,k)$: $A(n+1,k) = A(n,k) \cup \{e_{n+1}\}$

2.4 Update the missing event set $M(n,k)$: $M(n+1,k) = M(n,k)$

2.5 IF $M(n+1,k) \subseteq A(n+1,k)$ then Goto **3.** ELSE set $n \leftarrow n+1$ and Goto **2.1.**

3. TIME WARPING OPERATION:

3.1 Obtain all missing event lifetimes to resume sample path construction at state $\hat{x}_k$:

$$\hat{y}_i(k) = \begin{cases} v_i(\hat{s}_i^k + 1) & \text{for } i \in M(n+1,k) \\ \hat{y}_i(k-1) & \text{otherwise} \end{cases}$$

3.2 Use (3)–(8) to determine $\hat{e}_{k+1}, \hat{x}_{k+1}, \hat{t}_{k+1}, \hat{y}_i(k+1)$ for all $i \in \Gamma(\hat{x}_{k+1}) \cap (\Gamma(\hat{x}_k) - \{\hat{e}_{k+1}\})$, $\hat{s}_i^{k+1}$ for all $i \in \mathcal{E}$.

3.3 Discard all used event lifetimes:

$$\tilde{\mathbf{V}}_i(n+1,k+1) = \tilde{\mathbf{V}}_i(n+1,k) - v_i(\hat{s}_i^k + 1) \text{ for all } i \in M(n+1,k)$$

3.4 Update the available event set $A(n+1,k)$:

$$A(n+1,k+1) = A(n+1,k) - \left\{i:\ i \in M(n+1,k),\ \hat{s}_i^{k+1} = s_i^{n+1}\right\}$$

3.5 Update the missing event set $M(n+1,k)$:

$$M(n+1,k+1) = \Gamma(\hat{x}_{k+1}) - \left(\Gamma(\hat{x}_k) - \{\hat{e}_{k+1}\}\right)$$

3.6 IF $M(n+1,k+1) \subseteq A(n+1,k+1)$ then $k \leftarrow k+1$ and Goto **3.1.** ELSE $k \leftarrow k+1, n \leftarrow n+1$ and Goto **2.1.**

Remark. If *TWA* is used *on line*, step **2.1** is taken care of automatically by the actual system. The additional operations involved are steps **2.2-2.4** and checking the condition in step **2.5**. If the latter is satisfied then the time warping operation in step **3** is responsible for constructing a segment of the desired sample path for as many events as possible, depending on step **3.6**.

Clearly, the computational requirements of *TWA* are minimal (adding and subtracting elements to sequences, simple arithmetic, and checking condition (12)). Rather, it is the storage of additional information that constitutes the major cost of the algorithm. Next we analyze the algorithm in terms of its computational efficiency and applicability. Furthermore, in section 5 we describe extensions to *TWA* allowing us to relax the assumptions presented in section 2.

4. Algorithm Evaluation

The evaluation of the *TWA*, in terms of its computational efficiency as an estimation technique, depends on whether it is used *on line* or *off line*. In the former case, data from a DES are directly observed and processed to concurrently construct sample paths over a set of different parameter settings. Aside from the storage requirement for event lifetimes, the computational cost of the algorithm is minimal. Thus, while for DES with exponential event lifetime distributions the *ASA* and *SC* methodologies discussed in section 1 are clearly more efficient, the ability to handle non-Markovian DES on line comes with minimal additional cost.

In the *off-line* case, the data processed by the *TWA* are obtained from a simulation model of a DES. Once again, if the model is characterized by exponential event lifetime distributions, then the *ASA* and *SC* methodologies are much more efficient (see section 6.3). Let us therefore concentrate on a comparison of the *TWA* with repetitive simulation for each parameter value (i.e., "brute-force" simulation), which is the obvious alternative for non-Markovian systems. In this case, a basic first question is whether the *TWA* is indeed more efficient than brute-force simulation. Thus, if we let T_{BF} be the total simulation time (in CPU time units) required to generate the sample paths $\xi(\theta_1), \ldots, \xi(\theta_M)$ through M individual simulations (brute force simulation) and T_{CS} be the time required to generate the same sample paths through concurrent simulation, then the natural requirement is that

$$T_{CS} \leq T_{BF} \tag{16}$$

Observe that the operations in step **3.2** of *TWA* are the same as those in step **2.1**; these are required to update the state of the DES as described through (3)–(8) and must be also carried out when brute-force simulation is used. Therefore, this part is common to both brute-force simulation and *TWA*. The advantage of *TWA* as far as execution time is concerned is that no random variate generation is involved in any of the concurrently constructed sample paths. On the other hand, *TWA* introduces some overhead when writing to and reading from memory and when checking the subset condition in step **2.5**. As long as this overhead is less than the time taken to generate the random variates of the constructed sample path, (16) holds. In the next section, we proceed with a quantitative comparison between the *TWA* and brute-force simulation and define a convenient measure in order to accomplish this task.

4.1. The Speedup Factor

To define the *speedup factor* associated with concurrent simulation, suppose that the sample path constructed through our coupling approach were instead generated by a separate simulation whose length is defined by N total events. Let T_N be the time it takes (in CPU time units) to complete such a simulation run. Further, suppose that when the nominal simulation is executed with *TWA* as part of it, the total time is given by $T_N^o + \tau_K$, where T_N^o is the simulation time *without* the *TWA* and τ_K is the additional time involved in the concurrent construction of a sample path with $K \leq N$ events. We then define the *speedup*

factor due to *TWA* as

$$S = \frac{T_N/N}{\tau_K/K} \tag{17}$$

Thus, if a separate simulation (in addition to the one for the observed sample path) were to be used to generate a sample path under a new value of the parameter of interest, the computation time per event is T_N/N. If, instead, we use the coupling approach (i.e., *TWA*) in conjunction with the observed path, no such separate simulation is necessary, but the additional time per event imposed by the approach is τ_K/K, where $K \leq N$ in general. Clearly, $S \geq 1$ is required to satisfy the basic requirement expressed in (16).

4.2. An Upper Bound for Speedup

To determine an upper bound for the speedup factor S, first, note that the actual CPU time for a simulation consists of two components: *(a)* The time used to generate random variates for all distributions involved, and *(b)* The time used in all other simulation execution operations (updating the state, clock, and event list). Therefore we can write:

$$T_N = \alpha T_N + (1-\alpha)T_N \tag{18}$$

where T_N is the total CPU time to simulate N events in the nominal sample path, and α is the fraction of time used for generating random numbers and variates ($0 \leq \alpha \leq 1$). We assume α to be independent of N at steady state, which is true for sufficiently large values of N.

Now suppose that using the information obtained from the nominal sample path we construct a new sample path (under a different parameter setting) with $K \leq N$ events using the *TWA*. From the *TWA* it is clear that the CPU time to update the state, clock and event list in the nominal and perturbed sample paths are the same since steps **2.1** and **3.2** are identical. When simulating K events, this time is $(1-\alpha)T_K$. Furthermore, the overhead involved in implementing *TWA* consists of two parts: *(a)* The time taken to write to and read from memory, denoted by r_K, which depends on the number of random variates observed in the nominal sample path and used in the constructed sample path, and *(b)* The time taken to check the subset condition in step **2.5**, denoted by q_K. Note that the latter check takes place with every observed or constructed event. It follows that q_K is independent of α. The actual value of q_K depends on the cardinality of the set $M(n,k)$, denoted by $|M(n,k)|$. Therefore, since the *TWA* does not involve any random number generation, the total CPU time, τ_K, required to construct a sample path with K events is obtained by combining the second part of (18) with the read/write overhead component above:

$$\tau_K = (1-\alpha)T_K + r_K + q_K \tag{19}$$

Another way of viewing the total time τ_K is based on the following observation. When the *TWA* is executed along the nominal sample path, certain functions are performed whenever an event is observed (i.e., saving the event lifetime and checking (12)); therefore the total

time involved in such functions is independent of K and will be denoted by a. The remaining functions are performed whenever a constructed event is processed and the corresponding time is denoted by b. We can therefore write

$$\tau_K = a + bK \tag{20}$$

and it follows that $\tau_K/K = b + a/K$ is a decreasing function of K. Moreover, an upper bound for K is N, since the constructed sample path depends on the lifetimes observed in the nominal sample path; hence, $\tau_N/N \leq \tau_K/K$. We are now in a position to obtain an upper bound for the speedup factor as follows:

$$S = \frac{T_N/N}{\tau_K/K} \leq \frac{T_N}{\tau_N} = \frac{T_N}{(1-\alpha)T_N + r_N + q_N} \leq \frac{T_N}{(1-\alpha)T_N + r_N}. \tag{21}$$

As far as the last inequality is concerned, we have chosen to ignore q_N since, as will be shown in section 4.3, q_N is small when $|M(n,k)|$ is small. Finally, define

$$\beta = \frac{r_N}{T_N} \tag{22}$$

and substitute in (21) to get

$$S \leq \frac{T_N}{(1-\alpha)T_N + \beta T_N} = \frac{1}{1+\beta-\alpha} \tag{23}$$

This implies that the bound on the maximum achievable speedup is a function of the structural properties of the actual DES we simulate and the simulator used, manifested through α, and the specific processing environment used which determines β. Note also that the speedup as described in (21) is proportional to the ratio K/N, but for the calculation of the bound we assumed that this ratio takes its maximum value, i.e. $K/N = 1$. This implies that the larger the value of the ratio, the tighter the bound. On the other hand, note that it is possible that for certain systems this is not the case which implies that the observed speedup may be considerably less than the upper bound. In the worse case scenario, $K = 0$, which means that the speedup will also be zero.

Figure 2 shows some typical plots of the speedup upper bound as a function of α for various values of β. One can see that concurrent simulation is most beneficial when used to construct sample paths for DES that require a large number of random variates of complex distributions. More specifically, concurrent simulation is desirable when $\beta < \alpha$ where it is feasible to expect speedup factors greater than 1, i.e., concurrent simulation reduces the total simulation time. Note however, that $\beta < \alpha$ does not guarantee a speedup greater than 1 since the actual speedup includes q_K which has been ignored in the derivation of the bound (see (21)). Thus, the smaller q_K is, the tighter the bound. This motivates the discussion on regular DES presented next.

4.3. Regular Discrete Event Systems

As discussed in the previous section, systems with a small cardinality $|M(n,k)|$ for the set $M(n,k)$ attain higher speedup factors. Intuitively, systems with a large $|M(n,k)|$ imply

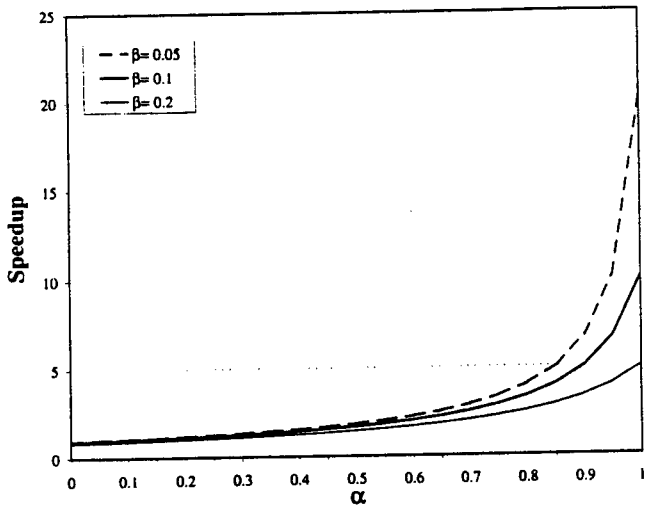


Figure 2. Speedup as a function of α.

that the simulator will spend a longer time updating the event list, which in turn implies that α is reduced, hence the speedup is also reduced as suggested by Figure 2. Thus, the role of $|M(n,k)|$ is important in our approach. The following result shows that $|M(n,k)|$ can be bounded by a quantity σ which depends entirely on the structural properties of a DES.

LEMMA 3 *Let $m(n,k) = |M(n,k)|$ be the cardinality of the set $M(n,k)$. Then,*

$$m(n,k) \leq \sigma + 1 \tag{24}$$

where σ is a non-negative integer such that:

$$\sigma = \max_k \{\sigma_k\} \qquad k = 1, 2, \ldots \tag{25}$$

$$\text{and } \sigma_k = |\Gamma(\hat{x}_k) - \Gamma(\hat{x}_{k-1})| \tag{26}$$

Proof: If $\hat{e}_k \notin \Gamma(\hat{x}_k)$, then, by the definition of $M(n,k)$ in (2), it is clear that $M(n,k) = \Gamma(\hat{x}_k) - \Gamma(\hat{x}_{k-1})$, therefore $|M(n,k)| = \sigma_k$. If, on the other hand, $\hat{e}_k \in \Gamma(\hat{x}_k)$, then $\hat{e}_k$ is an additional element that belongs to $M(n,k)$, i.e., $M(n,k) = (\Gamma(\hat{x}_k) - \Gamma(\hat{x}_{k-1})) \cup \{\hat{e}_k\}$, hence $|M(n,k)| = \sigma_k + 1$. Therefore, the result follows. ■

In simple terms, σ_k is the cardinality of a set that contains all events that are feasible at state $\hat{x}_k$ but which were not feasible in the previous state $\hat{x}_{k-1}$. It is important to observe that σ indeed depends entirely on the *structural properties* of a DES and is therefore not difficult to evaluate.

Clearly, the minimal value of σ is $\sigma = 0$. In this case, whenever an event occurs at any state, no new events are ever triggered. It is easy to see that this corresponds to a DES with $\Gamma(x) = \mathcal{E}$ for all states x, i.e., a system where all events are always feasible. Such DES are often of interest; however, they then trivially satisfy the constructability condition *(CO)*, therefore the *SC* or *ASA* techniques are preferable (see also Cassandras and Pan, 1995).

The next most desirable class of DES (in the sense of maximizing the speedup factor) is that characterized by $\sigma = 1$. This motivates the following

Definition. A DES is said to be *regular* if

$$\sigma = max\left\{|\Gamma(x') - \Gamma(x)| : x' = f(x, e), e \in \Gamma(x)\right\} \leq 1 \tag{27}$$

We refer to this as the *Regularity Condition* or condition **(R)**.

It turns out that regularity is not a particularly restrictive condition. For example, we show next that a large and particularly useful class of DES are regular.

PROPOSITION 1 *All DES represented through open or closed Jackson-like queueing networks satisfy the regularity condition.*

Proof: Consider a Jackson-like queueing network with n nodes and let $x = (x_1, \ldots, x_n)$ be a typical state, where x_i is the number of customers in the ith node. Let $\Gamma(x)$ be the feasible event set at that state. Every event in $\Gamma(x)$ is either an external arrival at some node k, denoted by a_k (if the system is open) or a departure from some node i, denoted by d_i. If there are multiple customer classes, each such event is further characterized by its corresponding class; however, as will be clear, the argument that follows is independent of the number of customer classes present. Denote the current state by x and the next state by $x' = f(x, e)$ such that e is either some arrival a_k or some departure d_k.

If a_k is the next event that occurs at state x then the following two cases are possible:

(i) if $x_k = 0$, then $\Gamma(x') = \Gamma(x) \cup \{d_k\}$ and therefore $|\Gamma(x') - \Gamma(x)| = 1$

(ii) if $x_k \geq 1$ then $\Gamma(x') = \Gamma(x)$ and therefore $|\Gamma(x') - \Gamma(x)| = 0$.

If, on the other hand, the next event is some d_i, then again two cases are possible:

First, if the departing customer leaves the system (if it is an open system), then, $\Gamma(x') = \Gamma(x) - \{d_i\}$ if $x_i' = 0$ and $\Gamma(x') = \Gamma(x)$ if $x_i' > 0$. Thus, $|\Gamma(x') - \Gamma(x)| = 0$.

Second, if the departing customer is routed to a node j, then, there are the following subcases:

(i) $x_i' = 0, x_j' = 1$. Then, $\Gamma(x') = (\Gamma(x) - \{d_i\}) \cup \{d_j\}$, and $|\Gamma(x') - \Gamma(x)| = 1$.

(ii) $x_i' = 0, x_j' > 1$. Then, $\Gamma(x') = \Gamma(x) - \{d_i\}$, and $|\Gamma(x') - \Gamma(x)| = 0$.

(iii) $x_i' > 0, x_j' = 1$. Then, $\Gamma(x') = \Gamma(x) \cup \{d_j\}$, and $|\Gamma(x') - \Gamma(x)| = 1$.

(iv) $x_i' > 0, x_j' > 1$. Then, $\Gamma(x') = \Gamma(x)$, and $|\Gamma(x') - \Gamma(x)| = 0$.

It follows that in all cases $|\Gamma(x') - \Gamma(x)| \leq 1$ and the proof is complete. ■

It is actually possible to incorporate features such as non-FIFO scheduling disciplines and preserve the regularity property of such DES. On the other hand, allowing for finite-capacity queues and "manufacturing blocking" generally increases the value of σ and violates regularity.

Example. Consider a $G/G/1/K$ system, with $x_k \in \{0, 1, \ldots, K\}$ and event set $\mathcal{E} = \{a, d\}$. If $x_k = 1$ and $x_{k-1} = 0$ (i.e., the kth event was an arrival), then $\sigma_k = |\{a, d\} - \{a\}| = |\{d\}| = 1$. For any other state transition, $x_k = x, x_{k-1} = x - 1$ (arrival), or $x_k = x - 1, x_{k-1} = x$ (departure), such that $x > 1$, $\sigma_k = |\Gamma(x_k) - \Gamma(x_{k-1})| = 0$. Therefore, $\sigma = \max_k\{\sigma_k\} = 1$. Note that if $K = \infty$, we still have $\sigma = 1$.

Example. In a serial network of two $G/G/1/K$ queues, let x_1, x_2 denote the number of customers in queue or server 1 and in queue or server 2 respectively. Assume that queue 1 has infinite capacity whereas queue 2 can only accommodate $K - 1$ customers. Then, $x_1 \in \{0, 1, \ldots\}$, $x_2 \in \{0, 1, \ldots, K\}$, and the event set is $\mathcal{E} = \{a, d_1, d_2\}$.

1. Under "communication blocking" (i.e., customers finding queue 2 full are lost):
   If $(x_1, x_2)_{k-1} = (0, 0)$ and $(x_1, x_2)_k = (1, 0)$, $\sigma_k = |\{a, d_1\} - \{a\}| = |\{d_1\}| = 1$.
   If $(x_1, x_2)_{k-1} = (1, 0)$ and $(x_1, x_2)_k = (0, 1)$, $\sigma_k = |\{a, d_2\} - \{a, d_1\}| = |\{d_2\}| = 1$.
   If $(x_1, x_2)_{k-1} = (0, 1)$ and $(x_1, x_2)_k = (1, 1)$, $\sigma_k = |\{a, d_1, d_2\} - \{a, d_2\}| = |\{d_1\}| = 1$.
   For all other state transitions $\sigma_k = 0$. Therefore, $\sigma = 1$.

2. Under "manufacturing blocking" (i.e., customers finding queue 2 full must wait in server 1 for an empty queueing slot), we introduce a third state variable $b \in \{0, 1\}$ such that $b = 1$ if a customer at server 1 is blocked and $b = 0$ otherwise.
   If $(x_1, x_2, b)_{k-1} = (0, 0, 0)$ and $(x_1, x_2, b)_k = (1, 0, 0)$, $\sigma_k = |\{a, d_1\} - \{a\}| = |\{d_1\}| = 1$.
   If $(x_1, x_2, b)_{k-1} = (1, 0, 0)$ and $(x_1, x_2, b)_k = (0, 1, 0)$, $\sigma_k = |\{a, d_2\} - \{a, d_1\}| = |\{d_2\}| = 1$.
   If $(x_1, x_2, b)_{k-1} = (0, 1, 0)$ and $(x_1, x_2, b)_k = (1, 1, 0)$, $\sigma_k = |\{a, d_1, d_2\} - \{a, d_2\}| = |\{d_1\}| = 1$.
   If $(x_1, x_2, b)_{k-1} = (x_1, K, 1)$ and $(x_1, x_2, b)_k = (x_1 - 1, K, 0)$, $\sigma_k = |\{a, d_1, d_2\} - \{a, d_2\}| = |\{d_1\}| = 1$.
   For all other state transitions $\sigma_k = 0$. Again, therefore, $\sigma = 1$. It is easy to check, however, that if a third server were introduced in this serial network, then we would have $\sigma > 1$.

5. Extensions of the TWA

In section 2 we stated a few assumptions that were made to simplify the development of our approach and keep the *TWA* notationally simple. It turns out that we can extend the application of *TWA* to DES by relaxing these assumptions at the expense of some extra work.

In *A2* we assumed that changing a parameter from θ_0 to some $\theta_m \neq \theta_0$ does not alter the event set $\mathcal{E}$. Clearly, if the new event set, $\mathcal{E}_m$ is such that $\mathcal{E}_m \subseteq \mathcal{E}$, the development and analysis of *TWA* is not affected. If, on the other hand, $\mathcal{E} \subset \mathcal{E}_m$, this implies that events required to cause state transitions under θ_m are unavailable in the observed sample path, which make the application of our algorithm impossible. In this case, one can introduce *phantom* event sources which generate all the unavailable events as described, for example, in Cassandras and Shi (1996), provided that the lifetime distributions of these events are known. The idea of phantom sources can also be applied to DES that do not satisfy *A1*. In this case, if a sample path remains suspended for a long period of time, then a phantom source can provide the required event(s) so that the sample path construction can resume.

In *A3* we assumed that changing a parameter from θ_0 to some $\theta_m \neq \theta_0$ does not affect the distribution of one or more event lifetime sequences. This assumption is used in (9) where the observed lifetime $v_i(s_i^n)$ is directly suffix-added to the sequence $\tilde{\mathbf{V}}_i(n+1,k)$. Note that this problem can be overcome by transforming observed lifetimes $\mathbf{V}_i = \{v_i(1), v_i(2), \ldots\}$ with an underlying distribution $G_i(\theta_0)$ into samples of a similar sequence corresponding to the new distribution $G_i(\theta_m)$ and then suffix-add them in $\tilde{\mathbf{V}}_i(n+1,k)$. This is indeed possible, if $G_i(\theta_0)$, $G_i(\theta_m)$ are known, at the expense of some additional computational cost for this transformation (for example, see Cassandras, 1993). One interesting special case arises when the parameter of interest is a scale parameter of some event lifetime distribution (e.g., it is the mean of a distribution in the Erlang family). Then, simple rescaling suffices to transform an observed lifetime v_i under θ_0 into a new lifetime $\hat{v}_i$ under θ_m:

$$\hat{v}_i = (\theta_m/\theta_0)v_i$$

Finally note that in a simulation environment it is possible to eliminate the overhead q_K which is due to checking the subset condition in step **2.5**. In order to achieve this we need to eliminate the coupling between the observed and constructed sample paths. Towards this goal, we can simulate the nominal sample but rather than disposing the event lifetimes we save them all in memory. Once the simulation is done, we simulate one by one all the perturbed sample paths exactly as we do with the brute force simulation scheme but rather than generating the required random variates we read them directly from the computer memory. In this way we trade off computer memory for higher speedup. A quantification of this tradeoff is the subject of ongoing research.

6. Simulation Results

Simulation can be used to readily verify that the *Time Warping Algorithm* generates sample paths identical to those generated by a separate simulation run with the same input. In what

Table 1. Speedup factors for *Systems 1-2.*

System	Speedup factor	comments
1	2.44	utilization 0.25
	2.20	utilization 0.5
	2.44	utilization 0.5 + rare events
	2.18	utilization 0.75
2	3.64	Erlang order 2
	7.73	Erlang order 5
	16.48	Erlang order 10
	27.00	Erlang order 20

follows, we shall focus on studying the speedup factors, as defined in (17), obtained for a variety of DES.

6.1. Speedup Factor for Several Systems

We briefly describe below each specific DES considered with the speedup factors obtained in a particular computer environment (486 PC) used in this study.

System 1: $M/M/1/K$ with Multiple Classes of Customers

This is a single-server queueing system serving various classes of customers on a First-In-First-Out (FIFO) basis. Systems with 2 to 11 classes were implemented and each class has exponential service and interarrival times. A performance analysis problem which is often of interest in such systems is estimating the blocking probability of each customer class as a function of the buffer size K. Several values of arrival and service rates were used so as to achieve different server utilizations, as shown in Table 1. In addition, experiments included a system where one of the classes had a very low arrival rate (100 times slower) in order to investigate the behavior of *TWA* when a constructed sample path may be suspended for a long time before it gets a lifetime that is missing.

System 2: $G/G/1/K$ with Multiple Classes of Customers

This is the same as *System 1* with two classes of customers, but the service and interarrival times are now obtained from an Erlang distribution of the same order so that the server utilization is 0.67. As seen in Table 1, the speedup factor increases with the order of the Erlang distribution; this is expected, since Erlang random variate generation is more complex than the exponential one, which in turn implies that α is increased.

Table 2. Speedup factors for *System 3*.

λ_1	μ_{11}	μ_{21}	λ_2	μ_{12}	μ_{22}	Speedup factor
1.0	4.0	4.0	1.0	4.0	4.0	2.63
1.0	3.0	3.0	1.0	3.0	4.0	2.58
1.0	4.0	4.0	1.0	4.0	3.0	2.63
1.0	3.0	3.0	1.0	3.0	3.0	2.25

Table 3. Speedup factors for *System 4*.

System	λ_1	μ_{11}	μ_{21}	μ_{31}	λ_2	μ_{12}	μ_{22}	μ_{32}	Speedup factor
manufacturing	1.0	5.0	7.0	12.0	1.0	5.0	7.0	12.0	2.70
blocking	1.0	3.0	3.0	3.0	1.0	5.0	7.0	9.0	2.67
	1.0	2.0	6.0	3.0	2.0	5.0	4.0	3.0	2.62
	3.0	20.0	4.0	4.0	3.0	20.0	4.0	4.0	2.41
communication	1.0	5.0	10.0	15.0	1.0	5.0	10.0	15.0	2.66
blocking	1.0	3.0	3.0	3.0	1.0	5.0	7.0	9.0	2.49
	1.0	2.0	5.0	3.0	3.0	5.0	7.0	5.0	2.52
	2.0	3.0	10.0	4.0	5.0	7.0	10.0	9.0	2.50

System 3: Two Queues in Series

This is a serial network of two $M/M/1/K$ queues with two classes of customers, that operates under "manufacturing blocking" i.e., customers finding queue 2 full wait in server 1 for the next available queueing slot. Both servers operate under a FIFO scheduling policy. The arrival process is Poisson and the two servers are exponential. The performance measure of this system is the throughput as a function of the buffer allocation, i.e. (K_1, K_2) subject to $K_1 + K_2 = K$ where K_1, K_2 are the number of buffers allocated to each server. Table 2 shows some typical speedup results obtained for different parameter settings. The arrival rate of class i is λ_i and its service rate at the jth server is μ_{ij} $(i, j = 1, 2)$.

System 4: Three Queues in Series

This is the same as *System 3* except we extend it to three queues. Two cases are considered: "manufacturing" blocking (in which case we have $\sigma = 2$ in (27) and the system is not regular) and "communication" blocking. (in which case $\sigma = 1$ and the system is regular). Typical results are shown in Table 3. One observation is that the lack of regularity had minimal effect on the speedup factor in this case. Another is that, comparing Tables 2 and 3, the increase in size of the system from two queues to three had minimal effect on the speedup factors observed.

Table 4. Speedup factors for *System 5*.

λ_1	μ_1	λ_2	μ_2	$SW_{1\to 2}$	$SW_{2\to 1}$	Speedup factor
1.0	2.0	1.0	3.0	0.2	0.4	2.51
1.0	2.5	1.0	5.0	1.0	0.5	1.91
1.0	2.0	1.0	2.0	0.5	0.0	1.74
1.0	2.0	1.0	2.0	0.0	0.0	2.11

System 5: Two Queues in Parallel with a Single Bulk-Service Server

A single server is servicing two classes of customers using a Round Robin scheduling policy. Both customer arrival processes are Poisson and each class is serviced in batches with exponential service times. Furthermore, there is a deterministic time delay every time the server switches from one class to the other, denoted by $SW_{1\to 2}$ and $SW_{2\to 1}$. The performance measure is the average system delay as a function of the batch size of each class of customers. Some representative results are shown in Table 4.

As already mentioned, the main cost involved in using *TWA* is the storage requirement for event lifetimes in $\bar{\mathbf{V}}_i(n,k)$ for all $i \in \mathcal{E}$. In fact, it is not possible to bound these sequences in our approach. In practice, this implies that some stored event lifetimes may eventually have to be discarded to avoid memory overflows.

The performance of *TWA* clearly depends on the DES considered. The results in Table 1 that correspond to system 2 show the correlation of the complexity of the random variate generation processes involved with the performance of *TWA*. Erlang random variates are complex and require CPU intensive operations to obtain, thus the percentage of time spent in generating random variates (i.e., α as defined in section 4.2) increases, which in turn increases the speedup obtained and therefore makes *TWA* a more attractive approach. Conversely, this approach is not recommended for systems with completely deterministic event processes.

Lastly, it is our observation that for the systems studied performance was quite sensitive to specific implementations and processing architectures. This dependence is captured by the ratio β defined in section 4.2 and it is shown in Figure 2 for $\alpha > 0.5$. Thus, the speedup factors presented in the tables above may be largely dependent on the computer environment and specific implementation of *TWA* one adopts.

6.2. *Statistical Significance of Estimates obtained through TWA*

As indicated earlier, the constructed sample path may remain suspended for extended periods of time while waiting for one or more of the missing events. This in turn, implies that while the length of the observed sample path (N) is long enough to guarantee that the observed measures are statistically significant, the length of the constructed sample path (K) may not become long enough to provide such accuracy. Figure 3 shows the ratio (K/N) for System 1 in section 6.1 when there are four classes of customers and the observed sample path has five buffer slots. First note that when all events occur with similar frequency, the K/N ratio

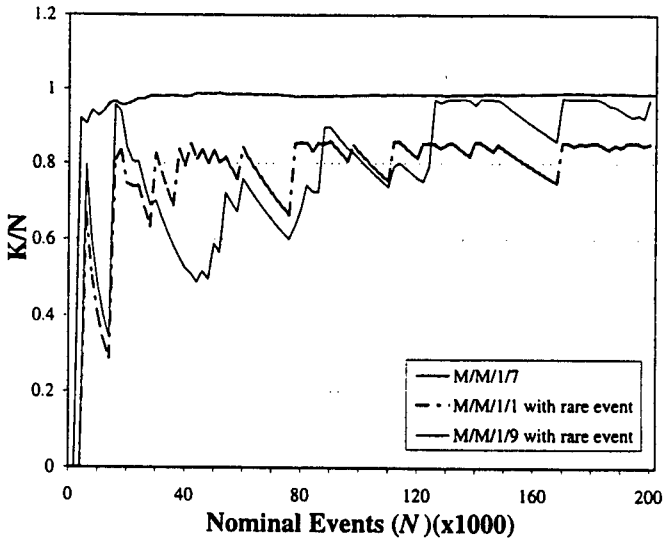


Figure 3. K/N ratio for system 1.

converges quickly ($M/M/1/7$ curve), whereas, when there is a rare event (class 4 arrival) often the constructed sample path is forced to wait for long intervals until the rare event is observed which causes the K/N ratio to become small. Once the missing event is observed, a large number of events may be immediately processed allowing K/N to increase again. This results in the initial large oscillations observed in Figure 3 which eventually diminish as N grows larger.

In addition, note that the parameter settings also affect the K/N ratio. In the case of a single buffer slot ($M/M/1/1$), the blocking probability is much larger than in the observed sample path, therefore, several observed departure events are not constructed because the corresponding customer was lost. For this reason, K/N converges to a value less than one (in this case 0.85). On the other hand, when the constructed sample path has nine slots, the observed and constructed sample paths have comparable blocking probabilities therefore most of the observed events are also constructed, so the K/N ratio is closer to one.

6.3. Speedup Comparisons

As mentioned earlier, *ASA* and *SC* are two methods that have been developed for constructing multiple sample paths at different parameter settings. In a simulation environment, both techniques can achieve higher speedup than the proposed *TWA*, as indicated in Figure 4 for an $M/M/1/K$ system studied over a range of values for K. *ASA* can achieve a speedup of up to 30, considerably higher than both *SC* and *TWA*, however, it is only applicable to systems with exponential event lifetime distributions (with the possibility of one non-Markovian event process, as noted in section 1). *SC* can achieve a speedup of up to 8, however it is

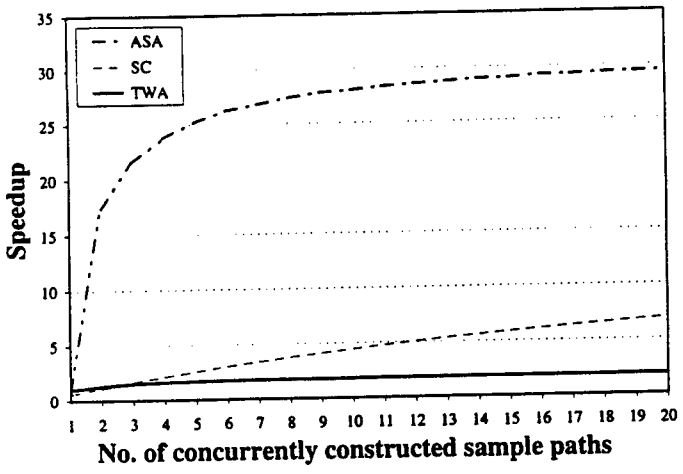


Figure 4. Speedup of *ASA*, *SC* and *TWA*, for an $M/M/1/K$ system.

also limited to exponential event lifetime distributions (unless approximations are used for systems with general distributions, as in Chen and Ho, 1995). In Figure 4, the speedup for the *TWA* turns out to be in the vicinity of 2.

Finally, note that from the definition of speedup (17), one would expect that it is not a function of the number of concurrently constructed sample paths. Intuitively, suppose that one is interested in concurrently constructing M sample paths. Therefore, using brute force it would take MT_N time units to generate MN events, while using any other constructability technique it would require $M\tau_K$ time units to construct MK events. Hence, the speedup factor should be independent of M. However, as described previously, the number of constructed events depends on the parameter settings as well. For this reason, the number of constructed events is given by $MK - K_0$ (not MK). Therefore,

$$S = \frac{MT_N/MN}{M\tau_K/(MK - K_0)} = \frac{T_N}{\tau_K}\left(\frac{K}{N} - \frac{K_0}{NM}\right)$$

which implies that it approaches a constant as M becomes larger.

7. Conclusions and Future Work

The sample path coupling approach we have presented is intended to solve the constructability problem described in section 2 without imposing any restrictions on the event processes in the DES as in earlier work. The approach leads to the specific "time warping" algorithm detailed in section 3.3, which was analyzed in terms of computational cost.

As pointed out in the Introduction, we emphasize once again that the proposed approach to the problem of constructability is suited to *on-line* sample path construction as a *concurrent estimation* scheme. In addition, it may be used as an off-line *concurrent simulation*

approach. In the former case, computational cost is not an issue, as the procedure we use involves only simple operations; rather, the storage requirement for observed event lifetimes becomes the limiting factor.

There remain a number of issues that require investigation in order to better assess the value of the concurrent sample path construction method we have presented. First, a detailed analysis of the extensions allowing us to relax Assumptions *(A1)*–*(A3)* as briefly outlined in section 5 is the subject of ongoing work. Second, the proposed sample path coupling approach is not limited to any specific performance measure of interest. However, one can expect that, depending on the nature of a performance measure to be estimated it should be possible to utilize only part of a constructed sample path, hence increasing the efficiency of the *TWA* in terms of speedup.

Finally, note that the proposed estimation technique, when used together with any optimization algorithm based on comparisons of the system's performance under different parameters, immediately improves the convergence rate of the algorithm because concurrent estimation/simulation inherently uses the *common random numbers (CRN)* scheme which has been observed experimentally and was proved theoretically in some cases to be effective in variance reduction (Dai and Chen, 1997).

Acknowledgements

This work was supported in part by the National Science Foundation under Grants EID-9212122 and EEC-9527422, by AFOSR under grants F49620-95-1-0131 and F49620-98-1-0387, and by the Air Force Rome Laboratory under contracts F30602-95-C-0242 and F30602-97-C-0125.

Notes

1. For readers familiar with the structure of a standard discrete event simulator, this idea applied to such a simulator implies the following: when an event in the "event calendar" is processed it is normally discarded; what is proposed here is that *it should not be discarded* but saved and used by concurrently generated sample paths in their respective event calendars.

References

Cassandras, C. G. 1993. *Discrete Event Systems, Modeling and Performance Analysis*. IRWIN.

Cassandras, C. G. and Julka, V. 1994. Descent algorithms for discrete resource allocation problems. *Proc. 33rd IEEE Conf. Decision and Control*, pp. 2639–2644.

Cassandras, C. G., Lee, J. I., and Ho, Y. C. 1990. Efficient parametric analysis of performance measures for communication networks. *IEEE Journal on Selected Areas in Communications* 8(9): 1709–1722.

Cassandras, C. G. and Pan, J. 1995. Parallel sample path generation for discrete event systems and the traffic smoothing problem. *Journal of Discrete Event Dynamic Systems* 5(2/3): 187–217.

Cassandras, C. G. and Panayiotou, C. G. 1996. Concurrent sample path analysis of discrete event systems. *Proceedings of the 35th Conference on Decision and Control*, pp. 3332–3337.

Cassandras, C. G. and Strickland, S. G. 1989a. On-line sensitivity analysis of Markov chains. *IEEE Transactions on Automatic Control* AC-34: 76–86.

Cassandras, C. G. and Strickland, S. G. 1989b. Observable augmented systems for sensitivity analysis of Markov and semi-Markov processes. *IEEE Transactions on Automatic Control* AC-34: 1026–1037.

Cassandras, C. G. and Shi, W. 1996. Perturbation analysis of multiclass multiobjective queueing systems with 'quality-of-service' guarantees. *Proceedings of the 35th Conference on Decision and Control*, pp. 3322–3327.

Chen,C. H., and Ho, Y. C. 1995. An approximation approach of the standard clock method for general discrete event simulation. *IEEE Transactions on Control Applications* 3: 309–317.

Dai, L., and Chen, C. H. 1997. Rates of convergence of ordinal comparison for dependent discrete event dynamic systems. *Journal on Optimization Theory and Applications* 94.

Glasserman, P. 1991. *Gradient Estimation via Perturbation Analysis*. Boston: Kluwer.

Glasserman, P. and Yao, D. 1994. *Monotone Structure in Discrete-Event Systems*. Wiley Interscience.

Gong, W. B., Ho, Y. C. and Zhai, W. 1992. Stochastic comparison algorithm for discrete optimization with estimation. *Proc. 31st IEEE Conf. Decision and Control*, pp. 795–802.

Ho, Y. C. and Cao, X. 1991. *Perturbation Analysis of Discrete Event Systems*. Boston: Kluwer.

Panayiotou, C. G. and Cassandras, C. G. 1996. Optimization of Kanban-based production systems. *Proceedings of WODES '96*, pp. 39–44.

Panayiotou, C. G. and Cassandras, C. G. 1997. Dynamic resource allocation in discrete event systems. *Proceedings of IEEE Mediterranean Conference on Control and Systems*.

Park, Y. and Chong, E. K. P. 1995. Distributed inversion in timed discrete event systems. *J. of Discrete Event Dynamic Systems* 5(2/3): 219–241.

Vakili, P. 1991. A standard clock technique for efficient simulation. *Operations Research Letters* 10: 445–452.

Yan, D. and Mukai, H. 1992. Stochastic discrete optimization. *SIAM J. on Control and Optimization* 30.

STOCHASTIC DISCRETE OPTIMIZATION USING A SURROGATE PROBLEM METHODOLOGY*

Kagan Gokbayrak
Department of Manufacturing Engineering
Boston University
Boston, MA 02215
kgokbayr@bu.edu

Christos G. Cassandras
Department of Manufacturing Engineering
Boston University
Boston, MA 02215
cgc@bu.edu

Abstract

We consider stochastic discrete optimization problems where the decision variables are non-negative integers. We propose and analyze an *on-line* control scheme which transforms the problem into a "surrogate" continuous optimization problem and proceeds to solve the latter using standard gradient-based approaches while simultaneously updating both actual and surrogate system states. Convergence of the proposed algorithm is established and it is shown that the discrete state neighborhood of the optimal surrogate state contains the optimal solution of the original problem. Numerical results are included in the paper illustrating the fast convergence properties of this approach.

1 Introduction

We consider stochastic discrete optimization problems where the decision variables are non-negative integers. In the context of resource allocation, for example, classic problems of this type include buffer allocation in queueing models of manufacturing systems or communication networks and transmission scheduling in radio networks. In the context of Discrete Event Systems (DES), integer-valued control variables have proven to be very common (e.g., as threshold parameters in many control policies), making the issue of optimizing over such variables of particular interest.

Let $r \in Z_+^N$ be the decision vector or "state". In general, there is a set of feasible states denoted by A_d such that $r \in A_d$ represents a constraint. For example, in a typical resource allocation problem, r_i denotes the number of resources that user i is assigned subject to a capacity constraint of the form $A_d = \{r : \sum_{i=1}^N r_i = K\}$. In a stochastic setting, let $L_d(r,\omega)$ be the cost incurred over a specific sample path (denoted by ω) and $J_d(r)$ be the expected cost of a system operating under r. Then, the discrete optimization problem we are interested in is the determination of $r^* \in A_d$ such that

$$J_d(r^*) = \min_{r \in A_d} J_d(r) = \min_{r \in A_d} E_\omega[L_d(r,\omega)] \qquad (1)$$

In general, this is a notoriously hard stochastic integer programming problem. Even in a deterministic setting, where we may set $J_d(r) = L_d(r,\omega)$, this class of problems is NP-hard (see [14] [12] and references therein). In some cases, depending upon the form of the objective function $J_d(r)$ (e.g., separability, convexity), efficient algorithms based on finite-stage dynamic programming or generalized Lagrange relaxation methods are known. Alternatively, if no a priori information is known about the structure of the problem, then some form of a search algorithm is employed (e.g., Simulated Annealing [1], Genetic Algorithms [11]). When the system operates in a stochastic environment (e.g., in a resource allocation setting users request resources at random time instants or hold a particular resource for a random period of time) and no closed-form expression for $E_\omega[L_d(r,\omega)]$ is available, the problem is further complicated by the need to estimate $E_\omega[L_d(r,\omega)]$. This generally requires Monte Carlo simulation or direct measurements made on the actual system.

While the area of stochastic optimization over *continuous* decision spaces is rich and usually involves gradient-based techniques as in several well-known stochastic approximation algorithms [13],[15], the literature in the area of *discrete* stochastic optimization is relatively limited. Most known approaches are based on some form of random search (e.g., [16],[8]) or, more recently, the use of the *ordinal* optimization approach presented in [9]. For a class of resource allocation problems of the form (1), an approach of this type was used in [3]. Even though the approach in [3] yields a fast resource allocation algorithm, it is still constrained to iterate so that every step involves the transfer of no more

*This work was supported in part by the National Science Foundation under Grants EEC-95-27422 and ACI-98-73339, by AFOSR under contract F49620-98-1-0387 and by the Air Force Research Laboratory under contract F30602-97-C-0125.

than a single resource from one user to some other user. One can expect, however, that much faster improvements can be realized in a scheme allowed to reallocate multiple resources from users whose cost-sensitivities are small to users whose sensitivities are much larger. With this motivation in mind, it is reasonable to pose the following question: Is it possible to transform a *discrete* optimization problem as in (1) into a "surrogate" *continuous* optimization problem, proceed to solve the latter using standard gradient-based approaches, and finally transform its solution into a solution of the original problem? Moreover, is it possible to design this process for *on-line* operation? That is, at every iteration step in the solution of the surrogate continuous optimization problem, is it possible to immediately transform the surrogate continuous state into a feasible discrete state r? This is crucial, since whatever information is used to drive the process (e.g., sensitivity estimates) can only be obtained from a sample path of the *actual* system operating under r.

In this paper, we transform the original discrete set A_d into a continuous set over which a "surrogate" optimization problem is defined and subsequently solved. As in earlier work in [3], [4] and unlike algorithms presented in [12], an important feature of our approach is that every state r in the optimization process remains feasible, so that our scheme can be used *on line* to adjust the decision vector as operating conditions (e.g., system parameters) change over time. Thus, at every step of the continuous optimization process, the continuous state obtained is mapped back into a feasible discrete state; based on a realization under this feasible state, new sensitivity estimates are obtained that drive the surrogate problem to yield the next continuous state. The proposed scheme, therefore, involves an interplay of sensitivity-driven iterations and continuous-to-discrete state transformations. The key issue then is to show that when (and if) an optimal allocation is obtained in the continuous state space, the transformed discrete state is in fact r^* in (1).

2 Basic approach

In the sequel, we shall adopt the following notational conventions. We shall use subscripts to indicate components of a vector (e.g., r_i is the ith component of r). We shall use superscripts to index vectors belonging to a particular set (e.g., r^j is the jth vector of the same form as r within a subset of A_d that contains such vectors). Finally, we reserve the index n as a subscript that denotes iteration steps and not vector components (e.g., r_n is the value of r at the nth step of an iterative scheme, not the nth component of r).

One common method for the solution of this problem is to relax the integer constraint on all r_i so that they can be regarded as continuous (real-valued) variables and then apply standard optimization techniques such as gradient-based algorithms. The resulting "surrogate" problem then is to find $\rho^* \in A_c$ so that

$$J_c(\rho^*) = \min_{\rho \in A_c} J_c(\rho) = \min_{\rho \in A_c} E_\omega[L_c(\rho, \omega)] \qquad (2)$$

where $\rho \in \mathbb{R}_+^N$ is a real-valued state, A_c is the convex hull of the original constraint set A_d, and $L_c(\rho, \omega)$ is the cost function over a specific sample path (denoted again by ω) when the state is ρ. Assuming an optimal solution ρ^* can be determined, this state must then be mapped back into a discrete vector by some means (usually, some form of truncation). Even if the final outcome of this process can recover the actual r^* in (1), this approach is strictly limited to *off-line* analysis: When an iterative scheme is used to solve the problem in (2) (as is usually the case except for very simple problems of limited interest), a sequence of points $\{\rho_n\}$ is generated; these points are generally continuous states in A_c, hence they may be infeasible in the original discrete optimization problem. Moreover, if one has to estimate $E_\omega[L_c(\rho, \omega)]$ or $\frac{\partial E_\omega[L_c(\rho,\omega)]}{\partial \rho}$ through simulation, then a simulated model of the surrogate problem must be created, which is also not generally feasible. If, on the other hand, the only cost information available is through direct observation of sample paths of an actual system, then there is no obvious way to estimate $E_\omega[L_c(\rho, \omega)]$ or $\frac{\partial E_\omega[L_c(\rho,\omega)]}{\partial \rho}$, since this applies to the real-valued ρ, not the actual cost observable under integer-valued r.

In this paper we propose a different approach intended to operate *on line*. In particular, we still invoke a relaxation such as the one above, i.e., we formulate a surrogate continuous optimization problem with some state space $A_c \subset \mathbb{R}_+^N$ and $A_d \subset A_c$. However, at every step n of the iteration scheme involved in solving the problem, the discrete state is updated through a mapping of the form $r_n = f_n(\rho_n)$ as ρ_n is updated using a stochastic approximation algorithm. This has two advantages: First, the cost of the original system is continuously adjusted (in contrast to an adjustment that would only be possible at the end of the surrogate minimization process); and second, it allows us to make use of information typically used to obtain cost sensitivities from the actual operating system at every step of the process. It is important to note that $\{r_n\}$ corresponds to feasible realizable states based on which one can evaluate sensitivity estimates from observable data, i.e., a sample path of the actual system under r_n (not the surrogate state ρ_n). We can therefore see that this scheme is intended to combine the advantages of a stochastic approximation type of algorithm with the ability to obtain sensitivity estimates with respect to discrete decision variables. In particular, sensitivity

estimation methods for discrete parameters based on Perturbation Analysis (PA) and Concurrent Estimation [10],[2] are ideally suited to meet this objective.

3 Continuous-to-discrete state transformations

In the sequel we will assume that $A_c \cap \mathbb{Z}^N = A_d$ where A_c is the convex hull of A_d; that is, all the *discrete* allocations contained in A_c are feasible. Given a vector $\rho \in \mathbb{R}_+^N$, we begin by specifying a set F_ρ of mappings $f(\rho)$. To do so, first define

$$I_\rho = \{i \,|\, \rho_i \in Z_+\} \tag{3}$$

to be the set of components of ρ (i.e., user indices) that are strictly integer. Next, define

$$\{f_i^+(\rho), f_i^-(\rho)\} = \begin{cases} \{\rho_i\} & \text{if } i \in I_\rho \\ \{\lceil \rho_i \rceil, \lfloor \rho_i \rfloor\} & \text{otherwise} \end{cases} \tag{4}$$

where, for any $x \in \mathbb{R}$, $\lceil x \rceil$ and $\lfloor x \rfloor$ denote the ceiling (smallest integer $\geq x$) and floor (largest integer $\leq x$) of x respectively. Then, let

$$F_\rho = \{f \,|\, f : A_c \to A_d, \forall i \; f_i(\rho) \in \{f_i^+(\rho),\; f_i^-(\rho)\}\}$$

$f \in F_\rho$ transforms continuous state vector $\rho \in A_c$ into a "neighboring" discrete state vector $r \in A_d$ obtained by seeking $\lceil \rho_i \rceil$ or $\lfloor \rho_i \rfloor$ for each component. Note that for all $f \in F_\rho$ and $r \in A_d$, $f(r) = r$.

Definition 1 *The set of all feasible discrete neighboring states of $\rho \in A_c$ is:*

$$\mathcal{N}(\rho) = \{r \,|\, r = f(\rho) \text{ for some } f \in F_\rho\} \tag{5}$$

Although much of the ensuing analysis applies to any constraint set A_c, we shall limit ourselves to the common case of a total resource capacity constraint:

$$A_c = \left\{\rho : \sum_{i=1}^N \rho_i = K\right\} \tag{6}$$

In this case, a more explicit and convenient characterization of the set $\mathcal{N}(\rho)$ is possible by defining the *residual* vector $\tilde{\rho} \in [0,1)^N$ of the continuous state ρ, given by $\tilde{\rho} = \rho - \lfloor \rho \rfloor$ where $\lfloor \rho \rfloor$ is the vector whose components are $\lfloor \rho \rfloor_i = \lfloor \rho_i \rfloor$. Then, set

$$m_\rho = \sum_{i=1}^N \tilde{\rho}_i = \sum_{i=1}^N (\rho_i - \lfloor \rho_i \rfloor) = K - \sum_{i=1}^N \lfloor \rho_i \rfloor \tag{7}$$

and note that $m_\rho \in \mathbb{Z}_+$ is an integer with the following convenient interpretation: If all users are assigned $\lfloor \rho_i \rfloor$ resources, then m_ρ is the total *residual resource capacity* to be allocated. Recalling the definition of the set I_ρ in (3), let $q = |I_\rho|$ be the number of components of ρ with strictly integer values. Then, $m_\rho \in \{0, \ldots, N-q-1\}$.

F_ρ may be interpreted as a set of mappings that allocate m_ρ residual resources over all $i \notin I_\rho$ in addition to a fixed integer $\lfloor \rho_i \rfloor$ already assigned to i. Let us then define $\tilde{r}^j(\rho) \in \{0,1\}^N$ to be the jth residual discrete vector corresponding to some given ρ which satisfies $\sum_{i=1}^N \tilde{r}_i^j = m_\rho$ and $\tilde{r}_i^j = 0$ for $i \in I_\rho$. Thus, $\tilde{r}^j(\rho)$ is an N-dimensional vector with components 0 or 1 summing up to m_ρ. It is easy to see that the number of such distinct vectors, and hence the cardinality of the set $\mathcal{N}(\rho)$, is $\binom{N-q}{m_\rho}$. It follows that we can write, for all $f^j \in F_\rho$,

$$f^j(\rho) = \lfloor \rho \rfloor + \tilde{r}^j(\rho) \tag{8}$$

The following theorem establishes the fact that any $\rho \in A_c$ can be expressed as a convex combination of points $r \in \mathcal{N}(\rho)$. All proofs are omitted but may be found in [7].

Theorem 3.1 *Any $\rho \in A_c$ is a convex combination of its discrete feasible neighboring states, i.e., there exists a vector α such that*

$$\rho = \sum_{j=1}^M \alpha_j r^j, \text{ with } \sum_{j=1}^M \alpha_j = 1, \; \alpha_j \geq 0 \; \forall j = 1, .., M$$

where $M = |\mathcal{N}(\rho)|$ and $r^j \in \mathcal{N}(\rho)$, $j = 1, \ldots, M$.

This result asserts that every $\rho \in A_c$ belongs to $conv(\mathcal{N}(\rho))$, the convex hull of the feasible neighboring state set $\mathcal{N}(\rho)$ defined in (5). We can further limit the set of states over which such a convex combination can be formed as follows.

Corollary 3.1 *Any $\rho \in A_c$ is a convex combination of at most $N-q$ discrete feasible neighboring states, i.e., there exists a vector α such that for all $j = 1, .., N$*

$$\rho = \sum_{j=1}^{N-q} \alpha_j r^j \quad \text{with} \quad \sum_{j=1}^{N-q} \alpha_j = 1, \quad \alpha_j \geq 0 \tag{9}$$

where $r^j \in \mathcal{N}(\rho)$, $j = 1, \ldots, |\mathcal{N}(\rho)|$, $q = |I_\rho|$.

Definition 2 *$\mathcal{N}_{N-q}(\rho)$ is a subset of $\mathcal{N}(\rho)$ that contains $N-q$ (with $q = |I_\rho|$) linearly independent discrete neighboring states whose convex hull includes ρ.*

The existence of this set is guaranteed by the previous corollary and it plays a crucial role in our approach, because the mapping $f_n(\rho_n)$ will be an element of $\mathcal{N}_{N-q}(\rho_n)$. Therefore, it is important to be able to identify $N-q$ elements of $\mathcal{N}(\rho)$ that satisfy (9), and

hence determine $\mathcal{N}_{N-q}(\rho_n)$. The Simplex Method of Linear Programming (LP) is suitable for this task.

Given this "reduced" set of discrete feasible neighbors of ρ, $\mathcal{N}_{N-q}(\rho)$, we restrict our original set of continuous-to-discrete transformations F_ρ to

$$\mathcal{F}_\rho = \{f : f(\rho) \in \mathcal{N}_{N-q}(\rho)\} \tag{10}$$

Note that when the continuous state is ρ_n, the continuous-to-discrete mapping f_n will be an element of $\mathcal{F}_{\rho_n}$.

4 Construction of surrogate cost functions and their gradients

In the sequel, 'ω' will be dropped from $L_d(r,\omega)$ and from the "surrogate" cost function $L_c(\rho,\omega)$ in (2). Moreover, unless otherwise noted, all costs will be over the same sample path. Since our approach is based on iterating over the continuous state $\rho \in A_c$, yet drive the iteration process with information involving $L_d(r)$ obtained from a sample path under r, we must establish a relationship between $L_d(r)$ and $L_c(\rho)$. The choice of $L_c(\rho)$ is rather flexible and may depend on information pertaining to a specific model and the nature of the given cost $L_d(r)$.

As seen in the previous section, it is possible that some components of ρ are integers, in which case the set I_ρ is non-empty and we have $q = |I_\rho| > 0$. In order to avoid the technical complications due to such integer components, let us agree that whenever this is the case we will perturb these components to obtain a new state $\hat{\rho}$ such that $I_{\hat{\rho}} = \emptyset$. In what follows, we will assume that all states ρ either have $I_\rho = \emptyset$ or have already been perturbed and relabeled ρ. Since q is going to be zero, we will rename $\mathcal{N}_{N-q}(\rho)$ as $\mathcal{N}_N(\rho)$.

We shall select a surrogate cost function $L_c(\rho)$ to satisfy the following two conditions:

(C1): *Consistency*: $L_c(r) = L_d(r)$ for all $r \in A_d$.

(C2): *Piecewise Linearity:* $L_c(\rho)$ is a linear function of ρ over $conv(\mathcal{N}_N(\rho))$.

Consistency is an obvious requirement for $L_c(\rho)$. Piecewise linearity is chosen for convenience, since manipulating linear functions over $conv(\mathcal{N}_N(\rho))$ simplifies analysis, as will become clear in the sequel. Given some state $\rho \in A_c$ and cost functions $L_d(r^j)$ for all $r^j \in \mathcal{N}_N(\rho)$, it follows from **(C2)** and (9) in Corollary 3.1 that we can write

$$L_c(\rho) = \sum_{j=1}^{N} \alpha_j L_c(r^j) \tag{11}$$

with $\sum_{j=1}^{N} \alpha_j = 1, \alpha_j \geq 0$ for all $j = 1,..,N$. Moreover, by **(C1)**, we have

$$L_c(\rho) = \sum_{j=1}^{N} \alpha_j L_d(r^j) \tag{12}$$

that is, $L_c(\rho)$ is a convex combination of the costs of N discrete feasible neighbors. Next, in order to use a stochastic approximation algorithm, we need sensitivity information provided through the sample gradient $\nabla L_c(\rho)$ expressed in terms of directly observable sample path data.

Since $L_c(\rho)$ is a linear function on the convex hull defined by the N discrete neighbors in (12), one can write

$$L_c(\rho) = \sum_{i=1}^{N} \beta_i \rho_i + \beta_0 \tag{13}$$

for some $\beta_i \in \mathbb{R}$, $i = 0,...,N$. Moreover, due to the linearity of $L_c(\rho)$ in $conv(\mathcal{N}_N(\rho))$, we have

$$\beta_i = \frac{\partial L_c}{\partial \rho_i},\ i = 1,...,N \tag{14}$$

For any discrete feasible neighboring state $r^j \in \mathcal{N}_N(\rho)$, one can use (13) and **(C1)** to write

$$L_d(r^j) = \sum_{i=1}^{N} \beta_i r_i^j + \beta_0,\ j = 1,\ldots,N \tag{15}$$

Letting $\nabla L_c(\rho)' = [\beta_1,\ldots,\beta_N]$ be the gradient of $L_c(\rho)$, our objective is to obtain an expression for $\beta_1,\ldots,\beta_N$ (not β_0) in terms of $L_d(r^j)$. Note that $L_d(r^j)$ are costs that can be evaluated at feasible states $r^j \in \mathcal{N}_N(\rho)$. These may be obtained by direct simulation; however, they are not available if a system is operating on line under one of these states, say r^1. This is where techniques such as Concurrent Estimation and Perturbation Analysis mentioned earlier can be used to facilitate this task.

To obtain expressions for $\beta_1,\ldots,\beta_N$ in terms of $L_d(r^j)$, let r^1 be the current state of the system (without loss of generality), and define

$$\delta_i^{j,1} = r_i^j - r_i^1 = \begin{cases} -1 & \text{if } r_i^1 > r_i^j \\ 1 & \text{if } r_i^1 < r_i^j \\ 0 & \text{otherwise} \end{cases} \tag{16}$$

and

$$\Delta L_{j,1} = L_d(r^j) - L_d(r^1) \tag{17}$$

for all $i = 1,\ldots,N$ and $j = 2,\ldots,N$. Using (15), the last equation can be rewritten as

$$\Delta L_{j,1} = \sum_{i=1}^{N} \beta_i (r_i^j - r_i^1) = \sum_{i=1}^{N} \beta_i \delta_i^{j,1} \tag{18}$$

If all $L_d(r^j)$ in (17) are observable, then (18) provides $N-1$ linearly independent equations for the N variables $\beta_1,\ldots,\beta_N$. An additional equation is obtained as follows: In the stochastic approximation algorithm

$$\rho_{n+1} = \pi_{n+1}[\rho_n - \eta_n \nabla L_c(\rho_n)] \qquad (19)$$

let $\bar{\rho}_{n+1} = \rho_n - \eta_n \nabla L_c(\rho_n)$ be an "intermediate" state prior to applying the projection π_{n+1}. In order to force $\bar{\rho}_{n+1}$ to satisfy the total capacity constraint (6), i.e.,

$$\sum_{i=1}^{N} (\bar{\rho}_{n+1})_i = \sum_{i=1}^{N} (\rho_n)_i - \eta_n \sum_{i=1}^{N} \frac{\partial L_c(\rho_n)}{\rho_i} = K \qquad (20)$$

we require that

$$\sum_{i=1}^{N} \frac{\partial L_c(\rho_n)}{\rho_i} = \sum_{i=1}^{N} \beta_i = 0 \qquad (21)$$

The combination of (18) and (21) provides N equations used to determine unique $\beta_1,\ldots,\beta_N$. Specifically, define the $(N-1)$-dimensional vector $\Delta L' = [\Delta L_{2,1},\ldots,\Delta L_{N,1}]$ whose components were defined in (17), and the $(N-1)\times N$ matrix $\mathbf{\Delta R} = [\delta^{2,1},\ldots,\delta^{N,1}]'$whose rows are the vectors $\delta^{j,1} = \left[\delta_1^{j,1},\cdots,\delta_N^{j,1}\right]'$ as defined in (16). We then get from (18) and (21)

$$\nabla L_c(\rho) = \begin{bmatrix} \mathbf{\Delta R} \\ e' \end{bmatrix}^{-1} \begin{bmatrix} \Delta L \\ 0 \end{bmatrix} \qquad (22)$$

Therefore, $\nabla L_c(\rho)$, the sample gradient to be used as an estimate of $\nabla J_c(\rho)$ in (2), is obtained through the N costs $L_d(r^1),\ldots,L_d(r^N)$. The sample path at our disposal corresponds to one of the state vectors, which we have taken to be $r^1 \in \mathcal{N}_N(\rho)$, so that $L_d(r^1)$ is observable; the remaining $N-1$ costs therefore need to be obtained by some other means. One possibility is to perform $N-1$ simulations, one for each of these states. This, however, is not attractive for an on-line methodology. Fortunately, there exist several techniques based on Perturbation Analysis (PA) [10],[2] or Concurrent Estimation [5], which are ideally suited for this purpose; that is, based on observations of a sample path under r^i, one can evaluate $L_d(r^j)$ for states $r^j \neq r^i$ with limited extra effort. The efficiency of these techniques depends on the nature of the system and cost function. Systems with separable cost functions, i.e.

$$L_d(r) = \sum_{i=1}^{N} L_{d,i}(r_i) \qquad (23)$$

is one area where PA techniques prove to be particularly efficient. In such systems, one can write

$$L_c(\rho) = L_d(r^1) + \sum_{i=1}^{N} \Delta L_{d,i}(r^1)\,|\rho_i - r_i^1| \qquad (24)$$

where $\Delta L_{d,i}(r^1)$ is the change in cost by adding or removing (depending on the sign of $\rho_i - r_i^1$) a resource from the ith user (see [7]).

5 Optimization Algorithm

Summarizing the results of the previous sections and combining them with the stochastic approximation algorithm, we obtain the following optimization algorithm for the solution of the basic problem in (1). After initializing $\rho_0 = r_0$, for each iteration $n = 0, 1, \ldots,$

1. Perturb ρ_n so that $I_{\rho_n} = \emptyset$.
2. Determine $\mathcal{N}(\rho_n)$ [using (4)-(5)].
3. Determine $\mathcal{N}_N(\rho_n)$ [using the Simplex Method].
4. Select $f_n \in \mathcal{F}_{\rho_n}$ such that $r_n = f_n(\rho_n) = \arg\min_{r\in\mathcal{N}_N(\rho_n)} \|r - \rho_n\|$.
5. Collect $L_d(r^i)$ for all $r^i \in \mathcal{N}_N(\rho_n)$ [using Concurrent Estimation or Perturbation Analysis].
6. Evaluate $\nabla L_c(\rho_n)$ [using (22)].
7. Update state: $\rho_{n+1} = \pi_{n+1}[\rho_n - \eta_n \nabla L_c(\rho_n)]$.
8. If some stopping condition is not satisfied, repeat steps for $n+1$. Else, set $\rho^* = \rho_{n+1}$.

We finally obtain r^* as one of the neighboring feasible states in the set $\mathcal{N}_N(\rho^*)$. It is shown in [7] using techniques similar to [6] that under certain conditions $\{\rho_n\}$, with $\rho_0 \in A_c$ (initial condition) arbitrary, converges to ρ^* with probability 1. The following result establishes that we can then determine $r^* \in A_d$ that solves the optimization problem (1).

Theorem 5.1 *Let ρ^* minimize $L_c(\rho)$ over A_c. Then, there exists a discrete feasible neighboring state $r^* \in \mathcal{N}_N(\rho^*)$ which minimizes $L_d(r)$ over A_d and satisfies $L_d(r^*) = L_c(\rho^*)$*

6 Numerical Example

We illustrate our approach by means of a stochastic optimization application for a classic problem in manufacturing systems. Consider a kanban-based manufacturing system where 15 kanban (resources) are allocated to 3 servers (users) in series. The objective is to find the optimal allocation r^* that minimizes the Average Cycle Time (ACT), defined as the time between two job completions at the last server (this is equivalent to a throughput maximization problem). The arrival process is Poisson with rate $\lambda = 1.6$. The service times of the servers are exponentially distributed with rates $\mu_1 = 2.0$, $\mu_2 = 1.6$, $\mu_3 = 3.0$. In this case, we chose the step size to be constant at $\eta = 100$, while the observation intervals are increased in length. The system

started with an initial allocation $r_0 = [3,5,7]'$ and the algorithm performed as follows:

# jobs	ρ'	r'	ACT
0	[2.80, 4.90, 7.30]	[3, 5, 7]	0.830316
100	[7.44, 2.69, 4.87]	[7, 3, 5]	0.743377
300	[6.79, 4.02, 4.19]	[7, 4, 4]	0.737032
600	[7.81, 3.60, 3.59]	[8, 4, 3]	0.738288
2100	[7.57, 4.64, 2.79]	[7, 5, 3]	0.720974
25300	[6.97, 5.64, 2.39]	[7, 6, 2]	0.724723
32500	[6.75, 5.56, 2.69]	[7, 5, 3]	0.720611
74100	[6.32, 5.37, 3.31]	[6, 6, 3]	0.712891
316000	[6.61, 5.52, 2.87]	[7, 5, 3]	0.722122

Due to noise, r oscillates between three allocations, namely $[7,5,3]'$, $[6,6,3]'$ and $[7,6,2]'$. Using brute-force simulation, it was determined that these are the best three allocations with corresponding performance values which are very close to each other.

7 Conclusions and Future Work

A key contribution of the proposed surrogate problem methodology is its *on-line* control nature, based on actual data from the underlying system. One can therefore see that this approach is intended to combine the advantages of a stochastic approximation type of algorithm with the ability to obtain sensitivity estimates with respect to discrete decision variables. This combination leads to very fast convergence to the optimal point, as illustrated in Section 6. It appears, therefore, feasible to apply this approach to problems with local extremal points by developing a procedure that allows the algorithm to operate from multiple initial states in an effort to determine a global optimum.

Two issues that are the subject of ongoing research are (a) application of this approach to systems with different types of constraint sets, other than the capacity constraint in (6), and (b) the effect of using the Simplex method to determine the set $\mathcal{N}_{N-q}(\rho)$ on the gradient estimation.

References

[1] Aarts, E. and J. Korst, *Simulated Annealing and Boltzmann Machines.* Wiley, 1989.

[2] Cassandras, C. G. *Discrete Event Systems: Modeling and Performance Analysis.* Irwin Publ., 1993.

[3] Cassandras, C. G., L. Dai, and C. G. Panayiotou, "Ordinal optimization for deterministic and stochastic resource allocation.," *IEEE Trans. Automatic Control*, vol. 43, no. 7, pp. 881–900, 1998.

[4] Cassandras, C. G. and V. Julka, "A new approach for some combinatorially hard stochastic optimization problems," *Proc. of 31st Annual Allerton Conference on Communication, Control, and Computing*, pp. 667–676, 1993.

[5] Cassandras, C. G. and C. G. Panayiotou, "Concurrent sample path analysis of discrete event systems," *Journal of Discrete Event Dynamic Systems: Theory and Applications*, vol. 9,2, pp. 171–195, 1999.

[6] Chong, E. K. P. and P. J. Ramadge, "Convergence of recursive optimization algorithms using ipa derivative estimates," *Journal of Discrete Event Dynamic Systems: Theory and Applications*, vol. 1, pp. 339–372, 1992.

[7] Gokbayrak, K. and C. G. Cassandras, "Stochastic discrete optimization using a surrogate problem methodology," 1999. Submitted.

[8] Gong, W. B., Y. C. Ho, and W. Zhai, "Stochastic comparison algorithm for discrete optimization with estimation," *Proc. of 31st IEEE Conf. on Decision and Control*, pp. 795–800, 1992.

[9] Ho, Y. C., R. S. Sreenivas, and P. Vakili, "Ordinal optimization in DEDS," *J. of Discrete Event Dynamic Systems: Theory and Applications*, vol. 2, pp. 61–88, 1992.

[10] Ho, Y. C. and X. Cao, *Perturbation Analysis of Discrete Event Dynamic Systems.* Kluwer Academic Publishers, 1991.

[11] Holland, J. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, 1975.

[12] Ibaraki, T. and N. Katoh, *Resource Allocation Problems: Algorithmic Approaches.* MIT Press, 1988.

[13] Kiefer, J. and J. Wolfowitz, "Stochastic estimation of the maximum of a regression function," *Annals of Mathematical Statistics*, vol. 23, pp. 462–466, 1952.

[14] Parker, R. and R. Rardin, *Discrete Optimization.* Inc, Boston: Academic Press, 1988.

[15] Robbins, H. and S. Monro, "A stochastic approximation method," *Annals of Mathematical Statistics*, vol. 22, pp. 400–407, 1951.

[16] Yan, D. and H. Mukai, "Stochastic discrete optimization," *SIAM Journal on Control and Optimization*, vol. 30, 1992.

Simulation Driven Metamodeling of Complex Systems Using Neural Networks[1]

C.G. Cassandras[1], W.-B. Gong[2], C. Liu[2], C.G. Panayiotou[2], D.L. Pepyne[2]

[1]Department of Manufacturing Engineering
Boston University
Boston, MA 02215

[2]Department of Electrical & Computer Engineering
University of Massachusetts
Amherst, MA 01003

ABSTRACT

Simulation of large complex systems for the purpose of evaluating performance and exploring alternatives is a computationally slow process, currently still out of the domain of real-time applications. To overcome this limitation, one approach is to obtain a "metamodel" of the system, i.e., a "surrogate" model which is computationally much faster than the simulator and yet is just as accurate. We describe the use of Neural Networks (NN) as metamodeling devices which may be trained to mimic the input-output behavior of a simulation model. In the case of Discrete Event System (DES) models, the process of collecting the simulation data needed to obtain a metamodel can also be significantly enhanced through Concurrent Estimation techniques which enable the extraction of information from a single simulation that would otherwise require multiple repeated simulations. We will present applications of two benchmark problems in the C^{3}I domain: A tactical electronic reconnaissance model describing the flight of a reconnaissance aircraft carrying a bearing angle measuring sensor over a radar field in order to detect ground-based radar sites; and an aircraft refueling and maintenance system as a component of a typical Air Tasking Order (ATO). A comparative analysis with alternative metamodeling approaches indicates that a NN captures significant nonlinearities in the behavior of complex systems that may otherwise not be accurately modeled.

Keywords: Metamodeling, neural networks, concurrent estimation, discrete event systems, decision making

1. INTRODUCTION

Simulation is widely recognized as one of the most versatile and general-purpose tools available today for modeling complex processes and solving problems in design, performance evaluation, decision making, and planning. This includes C^{3}I environments, where most problems confronted by designers and decision makers are of such complexity that their analysis and solution far surpass the scope of available analytical and numerical methods; this leaves simulation as the only alternative of "universal" applicability. The ultimate purpose of simulation is often system performance evaluation and optimization. Typically, this involves the use of simulation to explore a multitude of "what if" scenaria. However, simulation is notoriously computer time-consuming when it comes to parametric studies of system performance. Unless substantial speedup of the performance evaluation process can be achieved, systematic performance studies of most real-world problems are beyond reach, even with supercomputers.

One way to achieve a speedup is through metamodeling. The main idea of metamodeling is to extract as much information from simulation as possible and use it to build a *surrogate model* of the system of interest which is much simpler (yet accurate) to work with. This is essentially analogous to constructing a function $F(x_1,\ldots x_N)$ from observations of the simulation output under a few selected combinations of simulation inputs $x_1,\ldots,x_N$. The problem, of course, is that the actual function we are trying to approximate with $F(x_1,\ldots x_N)$ is unknown. The most common approach is to try and build a *polynomial* expression. This is often inadequate because if the actual curve includes sudden jumps and asymptotic behavior (which is often the case in practice), then polynomial fits to such curves are known to be poor. Thus, obtaining a metamodeling device of "universal" applicability, i.e., one capable of generating functions of virtually arbitrary complexity, is needed. This paper explores *neural networks* as offering this capability and is intended to investigate the advantages and limitations of this approach.

Training neural networks for function approximation tasks typically requires a large number of training pairs, each under a different scenario. The obvious way to collect N training pairs is to perform N separate simulations: one for each scenario. If a typical simulation run takes T time units, this procedure requires a total of NT time units. For Discrete Event System (DES) simulation models, *concurrent estimation* can be used. In concurrent estimation, the objective of collecting N training pairs is met by performing

---

[1] Authors may be contacted via e-mail at: cgc@enga.bu.edu or (gong, cliu, panayiot, pepyne)@ecs.umass.edu.

a single baseline simulation, but endowing it with the capability to generate $N-1$ additional simulations concurrently. This is accomplished by exploiting "intelligent data sharing" which results in a total simulation time of $(T+c) << NT$, where c represents the overhead corresponding to this data sharing. Using concurrent estimation, therefore, reduces the simulation time needed to collect the data needed to train a neural network metamodel.

The remainder of this paper is organized as follows. Since our purpose is to compare polynomial metamodels to neural network metamodels, Section 2 describes the basics of polynomial and neural network curve fitting. Section 2 also briefly reviews concurrent estimation as a way to speed the process of collecting the simulation data needed to construct metamodels. Section 3 describes the two simulators that will be used to compare the two metamodeling approaches. Section 4 presents numerical comparison results. The paper ends in Section 5 with a conclusion and a discussion of open issues and ongoing work.

2. NEURAL NETWORKS FOR METAMODELING

For applications involving simulation, the time required to run the simulation may be very long or it may be necessary to perform many simulation runs. Metamodeling addresses these issues: If one can develop a metamodel that captures the functional input/output relationships embodied by the simulator, then it is possible to obtain the simulation data quickly and efficiently.

2.1. Polynomial Metamodels

Traditionally, polynomials have been used as metamodels. The functional mapping represented by a polynomial is determined by its order and the values of its coefficients. As an example, consider the 2-input, 1-output, second order polynomial, $\hat{y} = b_0 + b_1x_1 + b_2x_2 + b_3x_1x_2 + b_4x_1^2 + b_5x_2^2$ and suppose it is to be used to approximate some unknown function $y = f(x_1,x_2)$. The *least squares* approach gives a way to determine the coefficients which will minimize the mean squared error between the output of the polynomial and the output of the function being approximated. In the above example, there are 6 coefficients to determine. Thus, *at least* 6 experiments must be conducted. Suppose data has been collected for $N \geq 6$ such experiments. Index each experiment as $n = 0,...N$, and designate the inputs for the n-th experiment as $x_1(n)$ and $x_2(n)$, the output of the polynomial as $\hat{y}(n)$, and the target output of the function being approximated as $y(n)$. Next define $\hat{Y} = [\hat{y}(1), \hat{y}(2),..., \hat{y}(N)]^T$ as the vector of polynomial outputs, $Y = [y(1), y(2),..., y(N)]^T$ as the vector of target outputs, $E = Y - \hat{Y}$ as the error between the target output and the polynomial output, and $B = [b_0, b_1,..., b_5]^T$ as the vector of polynomial coefficients to be determined. The objective of the least squares approach then is to determine the values of the coefficients which will minimize the mean squared error, i.e., $\min_B [E^T E]$[1]. The solution to this problem is given by, $B^* = (X^T X)^{-1} X^T Y$ where X is the regressor matrix defined by,

$$X = \begin{bmatrix} 1 & x_1(1) & x_2(1) & x_1(1)x_2(1) & x_1^2(1) & x_2^2(1) \\ 1 & x_1(2) & x_2(2) & x_1(2)x_2(2) & x_1^2(2) & x_2^2(2) \\ & & & \cdots & & \\ 1 & x_1(N) & x_2(N) & x_1(N)x_2(N) & x_1^2(N) & x_2^2(N) \end{bmatrix}$$

Note, the determination of the coefficients B requires inverting the matrix $(X^T X)$. The size of this matrix is determined by the number of coefficients which is given by the number of inputs and the order of the polynomial. Since it can be difficult to invert large matrices, the least squares approach does not scale well to problems with many inputs and high order polynomials. In addition, the N inputs points cannot be chosen haphazardly lest the matrix be poorly conditioned or , even worse, singular.

As with all function approximation methods, the quality of the approximator depends on the particular set of N input points used to obtain the coefficients. Since, it is assumed that little is known about the function being approximated, selecting the best set of input points can be difficult. Strategies for choosing input points fall under the general heading of *experimental design*. For first and second order polynomials there are well established experimental designs for choosing the input points. For first order polynomials, the most popular are the *orthogonal designs*, so called because they result in a diagonal $(X^T X)$ [2]. These designs are useful because they minimize the variance of the coefficients when the function being approximated is stochastic. For second order polynomials, the most common design is the *central composite design* (CCD)[2]. For higher order polynomials, design methods are not as well established, but one common approach is to use *layered* CCD's.

2.2. Neural Network Metamodels

It is well known that multilayer feedforward neural networks are universal function approximators[3]. As such, they are often used for non-parametric modeling, and are well suited to the task of metamodeling. Fig. 2.1 depicts the architecture of a typical multilayer feedforward neural network. In this particular figure there are four layers, including one input layer, one output layer, and two

intermediate layers called the "hidden layers." Each layer consists of many nonlinear devices called "neurons." The output of each neuron, V_i, is called its "activation", and is a nonlinear function of a weighted sum of the inputs to the neuron minus a threshold, θ_i,

$$activation = g\left(\sum_{inputs} weights \times inputs - threshold\right)$$

The form of the function $g(\cdot)$ above is very important to the operation of a neural network. It is usually chosen to have the form of $g(x) = tanh(\beta x)$ and is called a "sigmoid function." As seen in the figure, the neurons are connected through links with different coefficients associated with each link. These coefficients are the "weights" in the above equation. It can be shown that if there are enough neurons in the hidden layers then there exists a set of weights that can approximate any function with a finite number of discontinuities to any desired degree of accuracy[3].

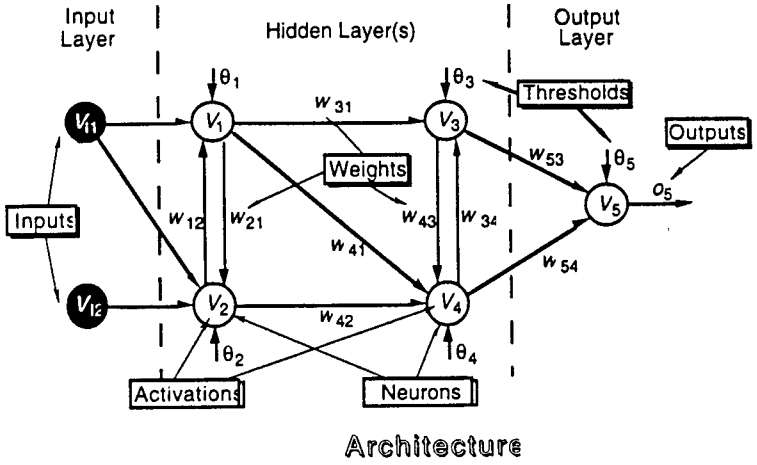


Figure 2.1. Architecture of a typical multilayer feedforward neural network.

Typically, the weights are adjusted using a "training" algorithm. The usual training objective is to adjust the weights to minimize the mean squared error between a desired target output and the network output for a specific set of inputs. To do this, a special algorithm known as the "backpropagation algorithm" is often used. The backpropagation algorithm, although quite ingenious, is really nothing more than an application of the chain rule of calculus to effect a gradient descent in the weight space. For a representative description of the algorithm see[4].

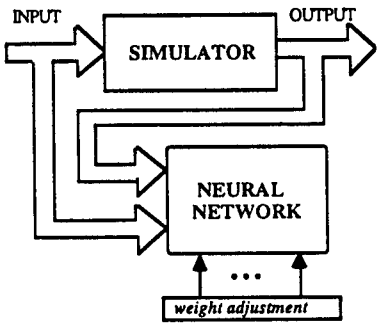


Figure 2.2. Neural network training through simulation to create a surrogate model.

The process of constructing a metamodel using a neural network is illustrated in Fig. 2.2. First, a large-scale simulation is executed. While the simulator is running, the neural network observes the simulator inputs and generates its own prediction of what the simulator output will be. It compares its output to the simulator output and uses the error to adjust its weights. In this way the neural network "learns" the input-output behavior of the simulator. One can visualize the neural network as an "entity" which is highly "intelligent" but has no knowledge of anything initially to apply its intelligence to. As it observes the simulation unfold, however, it learns from the basic cause-effect (i.e., input-output) relationships it observes. In fact, all the neural network does is adjust its weights so as to emulate the behavior it observes as closely as possible. When the training is done, the simulator can be taken away. The neural network is now a surrogate model: we give it some inputs (as if we were giving them to the simulator) and it immediately gives an output (as the simulator would).

One difficulty when using neural networks is choosing the number of neurons to use in the hidden layers. Since neural networks approximate functions by combining sigmoids (i.e., hidden layer neurons), the number of hidden layer neurons determines the ultimate complexity of the functions the network will be able to approximate. Too few and the network will underfit the training data, too many and it will overfit the data. In either case, the resulting performance will be less than satisfactory. To address this issue, we use the *Cascade Correlation Neural Network* (CCNN), which is a type of multilayer feedforward neural network that adds hidden layer neurons as it learns[5]. The CCNN starts with a single hidden layer neuron and adds more as needed during training which usually leads, not only to faster learning, but improved generalization performance. A detailed description of the CCNN training algorithm used for these metamodeling studies can be found in Cassandras and Gong[6].

The main advantages of neural networks over other approaches for function approximation (e.g., polynomials or rational functions) include: (i) *Generality* Neural networks are capable of approximating virtually any function. Polynomials and rational functions, in contrast, have known limitations in terms of their approximation capabilities. (ii) *Scalability* Neural networks scale easily as the problem size and complexity increases. When the system dimension is in the hundreds or thousands, mathematical formulae simply become too complicated to use. In fact, determining the coefficients for a polynomial or rational function can be completely infeasible when the dimension is large. (iii) *Inherent Parallelism*: Although the number of weights increases with the number of inputs and with the number of hidden neurons, the increased computational burden required to train the network can be met by exploiting the *inherent parallelism* of neural networks (i.e., each neuron can be implemented on a different processor). They are, therefore, ideal for modeling large-scale systems. (iv) *Well-suited for model sensitivity analysis*: It is always possible to do model sensitivity analysis with a trained neural network, as long as the related factors have been chosen as the inputs to the neural network. Thus, there is an excellent opportunity here to combine this metamodeling approach with concurrent/parallel techniques to perform model sensitivity analysis on the neural network surrogate model.

2.3. Concurrent Estimation

Constructing a neural network metamodel will usually require a large number of training points to achieve adequate performance. For Discrete Event System (DES) simulations, a technique called *Concurrent Estimation* can reduce the simulation time required to collect the necessary training data. The goal of concurrent estimation is the following: *From a **single** simulation, obtain performance results for several different values of the inputs.* The main idea behind the approach is to observe the evolution of a single (nominal) sample path of a DES as it operates under some parameter. As the nominal sample path evolves, observed data (e.g., event occurrences and their corresponding occurrence times) are processed to concurrently construct the set of sample paths that would have resulted if the system had operated under a set of different (hypothetical) parameters. Using these "concurrently constructed" sample paths, it is possible to "concurrently estimate" the corresponding performance measures. Thus, from a single simulation run training data can be collected which would otherwise require multiple runs to collect, therefore speeding up the process of data collection.

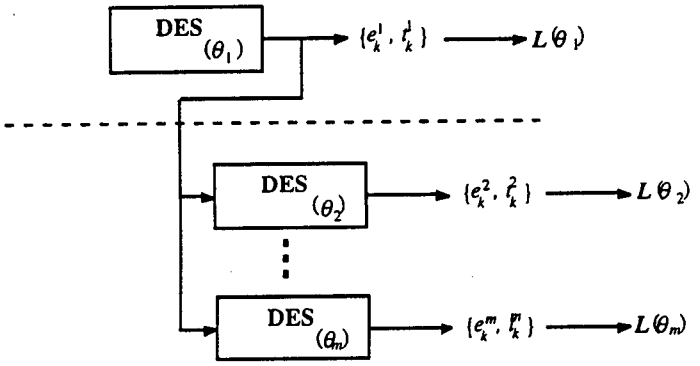


Figure 2.3. Concurrent Estimation.

To explain the essentials of concurrent estimation, consider a DES and a finite discrete parameter set $\Theta = \{\theta_1,\ldots,\theta_m\}$, where each parameter $\theta_j \in \Theta$, $j = 1,\ldots,m$ is in general vector-valued. Suppose the sample path generated by the DES is a function of the parameter θ_j, and designate the sample path generated under parameter θ_j by the sequence of pairs $\{e_k^j, t_k^j\}$, where $k = 1,2,\ldots$ is an event-counting index, e_k is the k-th event, and t_k is the occurrence time of the k-th event. Now, assume that the DES is operating under θ_1 and that all events and event times e_k^1, t_k^1 for $k = 1,2,\ldots$, are directly observable. The problem, then, is to use the

observations of the sample path $\{e_k^1, t_k^1\}$ to construct the sample paths $\{e_k^j, t_k^j\}$, $k = 1,2,\ldots$, for any θ_j, $j = 2,\ldots,m$. This problem of concurrently constructing multiple sample paths is referred to as the *sample path constructability problem*[7]. Note, any sample performance metric $L(\theta_j)$ is obtained as a function of the corresponding sample path $\{e_k^j, t_k^j\}$, $k = 1,2,\ldots$ as shown in Fig. 2.3. Here we are careful to distinguish between $L(\theta)$, the performance obtained over a *specific sample path* of the system and $E[L(\theta)]$, the *expectation over all possible sample paths*. The solution to the sample path constructability problem, if it exists, therefore enables us to learn about the behavior of a DES under all possible parameter values in Θ from a single "trial", i.e., a single sample path obtained under one parameter value. For DES in which all event processes are Markovian (memoryless), the Standard Clock scheme[8], and Augmented System Analysis (ASA)[9,10] provide two very efficient ways to obtain concurrent estimates. The Time Warping Algorithm (TWA)[11], while not as efficient as the other two, is a general-purpose scheme for DES with arbitrary event lifetime distributions.

A measure of the effectiveness of a concurrent estimation scheme is the "speedup factor" measuring how much faster one can obtain performance information for N concurrently constructed sample paths compared to N individual "brute force" simulation experiments. Typical numerical results for ASA give speedup factors of the order of 100, while for TWA factors of 2-20 or more are common[7,11] (the more complex the event lifetime probability distributions in a model, the greater the speedup). It is worth pointing out that if a parallel processing environment is used, the speedup factor becomes much greater by several orders of magnitude.

3. TWO BENCHMARK SIMULATION MODELS

To evaluate the efficacy of using the CCNN for metamodeling we performed numerical experiments using two different models, a Tactical Electronic Reconnaissance Simulation (TERSM) describing the flight of a reconnaissance aircraft carrying a bearing angle measuring sensor over a radar field in an effort to locate ground-based radar sites, and an Aircraft Refueling and Maintenance System (ARMS) which models a component of a typical Air Tasking Order (ATO).

3.1. Tactical Electronic Reconnaissance Simulator (TERSM)

TERSM, developed by the RAND corporation for the United States Air Force[12], is a very complex simulator that models the flight of a reconnaissance aircraft carrying a bearing angle measuring sensor over a radar field. As shown in Fig. 3.1, the aircraft flies with a fixed heading at a constant altitude and a constant velocity over a battlefield which contains many ground-based radar sites (emitters). As the aircraft flies, the sensor records bearing angle measurements of the emitters it detects and builds a list containing the circular error probable (CEP) for each one. The CEP is a disk of such size that the probability that the emitter is inside the disk is 50%. The simulator was originally used to compare competing sensors when making purchase decisions. Recently, TERSM has been used to develop and evaluate polynomial metamodels[13,14,15]. TERSM, therefore, provides an excellent testbed for comparing competing metamodeling approaches.

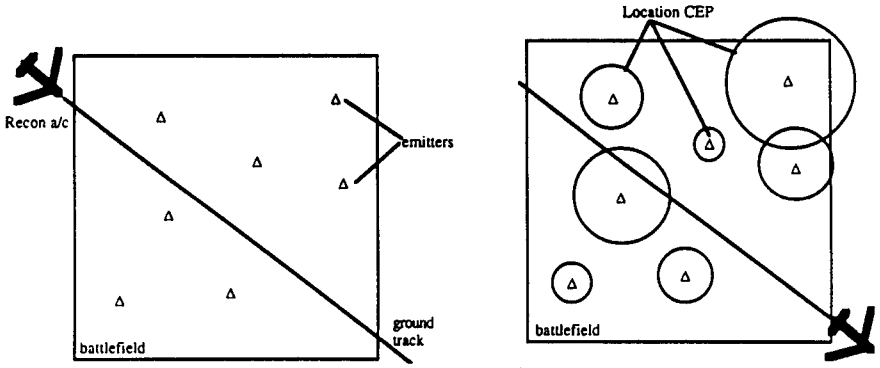


Figure 3.1. TERSM, start of mission (left), end of mission (right).

The operation of TERSM is similar in some respects to the familiar police scanner that one can buy in an electronics store. Like a police scanner, the sensor scans a set of frequency channels in some preprogrammed sequence. It dwells on a channel for a short period of time waiting for an emitter to transmit on that channel. If an emitter is transmitting, a series of sophisticated tests are performed to check if the sensor can detect it. The emitter must be within the line of sight of the sensor and not blocked by terrain or the curvature of the earth. The emitter must be within the field of view of the sensor's antenna pattern. The signal received by the sensor must be strong enough to detect, but not so strong that it exceeds the sensor's dynamic range. If all these tests pass, the sensor records the

detection data. After a short dwell time, the sensor switches to another channel. During the brief time period required to switch from one channel to the next, the sensor processes all the detections made on the previous channel to extract bearing angle measurements. The number of detections that can be processed during the channel switching time is determined by the channel capacity. TERSM has many inputs and is capable of generating many outputs. The inputs concern the aircraft flight data, the sensor data, and data for each emitter. The primary outputs are the number of emitters detected and the CEP for each one. Complete details describing TERSM can be found in[12].

3.2. Aircraft Refueling and Maintenance System

Although TERSM provides an excellent testbed to study the feasibility of the CCNN metamodeling approach, it lacks some of the features that constitute real challenges to metamodeling. For one, TERSM lacks significant randomness. That is, although the number of emitters detected does change as a function of the initial random number seed, the change is not significant. For another, TERSM does not exhibit asymptotic relationships which are very common in practice. Exposing such asymptotic behaviors often requires very long simulation runs. Avoiding long simulation runs is one of the real benefits of metamodeling. TERSM, on the other hand, has a running time of just under a minute on a modern workstation. For such a simulation, the benefits of metamodeling are not so compelling. Additionally, asymptotic behavior can be very difficult for polynomials to capture, and provides one of the primary motivations for the use of more powerful metamodeling methods like the CCNN.

We have therefore concentrated on identifying a good benchmark problem with the above features, based on the literature of military models[16,17]. In [17], for example, a model is considered for the process of moving and reassigning strategic airlift pilots with the objective of managing and ultimately minimizing moving costs while maintaining mission capability. Motivated by this problem, we have concentrated on an *Aircraft Refueling and Maintenance System* (ARMS). The basic ARMS model is shown in Fig. 3.2. As illustrated, ARMS is a multiclass queueing system. Jobs from each class $n = 1,...,N$ arrive with average rates λ_n to separate arrival queues with capacities C_n (possibly infinite). The system has θ tokens. At block 1, the jobs compete for a token on a priority basis, with the class 1 jobs having the highest priority. The job waiting in the highest priority arrival queue will be the first to get a token when one becomes available. After getting a token, the jobs enter a service queue with capacity C_s. At block 2 the jobs are selected from the service queue according to some service discipline (e.g., first in first out (FIFO)) and routed to one of $k = 1,...,K$ servers. The time to service a job is a random variable $\mu(n,k)$ which can vary as a function of the job class n and the particular server k. Upon completing service, the jobs proceed to block 3, where the token is returned to a token pool, and the job leaves the system.

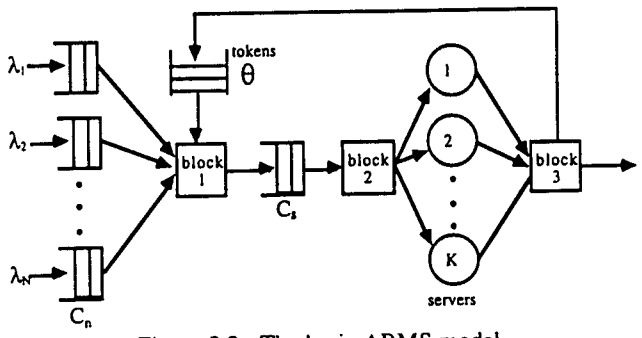


Figure 3.2. The basic ARMS model.

The basic ARMS model is very general and can be used to represent a large variety of Air Force C³I operations. For example, such a model can be used to represent computer networks, communications systems, or logistics problems. The specific problem we will consider is the aircraft refueling and maintenance system (ARMS). The ARMS problem has aircraft requesting to land at a particular site (airport or aircraft carrier) for refueling and/or maintenance purposes. Depending on aircraft type, a priority is assigned to each aircraft so that high-priority ones are served first. Since landing capacity and associated maintenance resources are limited, a specific number of "permits" (i.e., tokens) are available. An aircraft is, therefore, forced to wait until it receives a permit. Upon receiving a permit, the aircraft is guided to a refueling/maintenance area. If the resources required to complete the refueling/maintenance process are not immediately available (e.g., personnel, tools, spare parts, fuel), the aircraft is further delayed. When the aircraft completes service, the permit is returned to the permit pool, and the aircraft proceeds to take off and return to action. In studying this system, one is interested in minimizing the expected "down time" of an aircraft, with more emphasis given to certain types of aircraft (the ones given higher priority). At the same time, one is interested in keeping service costs within acceptable levels. From a modeling

standpoint, one must therefore determine functional relationships such as the expected down time of a priority 1 aircraft with respect to factors such as the number of permits; or the number of maintenance resources allocated to the refueling/maintenance process.

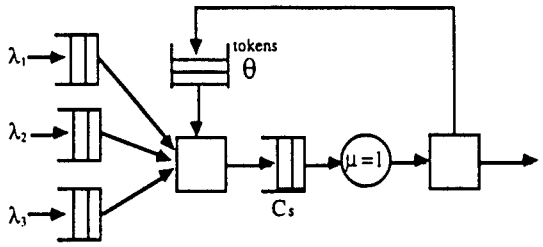


Figure 3.3. System used for metamodeling.

Our metamodeling studies in this paper focus on the 3-class, single server ARMS model shown in Fig. 3.3. In this model, the class 1 jobs have the highest priority, and the class 3 jobs the lowest. Job arrivals are Poisson with rates λ_i, where i is the customer class. In order for a job to be served, it must have one of θ tokens. Jobs with tokens queue up to be served by a single server. At the completion of service, the jobs leave the system, and the tokens are returned to the token pool for reuse. When different classes of jobs are competing for tokens, the class with the highest priority gets one first. Jobs in the same class compete for tokens on a first come, first served (FCFS) basis. The arrival queues have infinite capacity, and the server queue has capacity $C_s = \theta$. Jobs in the server queue are served FIFO (with no distinction made between jobs from different classes). Service is nonpreemptive (once a job begins service, service cannot be interrupted, and will continue until completion), and the service time is an exponential random variable with parameter $\mu_i = 1$, $i = 1,2,3$.

4. NUMERICAL EXPERIMENTS

This section gives numerical results to compare polynomial metamodels to CCNN metamodels for TERSM and ARMS. For the results that follow, the performance measure used is the *Mean Squared Error* (MSE) between the output of the simulator (TERSM or ARMS) and the corresponding output of the metamodel (polynomial or CCNN). For each input/output pair $i = 1,...,n$ in the data set, the mean squared error is given by,

$$MSE = \frac{1}{n}\sum_{i=1}^{n} e_i^2$$

where e_i is the error or difference between the output simulator and the output of the metamodel.

4.1. Baseline TERSM problem.

The baseline TERSM problem focused on the relationship between the number of emitters detected which have a CEP less than 5 nautical miles in radius and the following four inputs: the altitude of the aircraft (it flies at a constant altitude for the duration of its mission), the velocity of the aircraft (it flies at a constant velocity for the duration of the mission), the azimuth of the sensor (the angle of the sensor boresight relative to the aircraft heading), and the channel capacity of the sensor (the number of bearing angle measurements that can be processed during the channel switching time). This four input single output example (see Fig. 4.1) has been used in the literature to develop polynomial metamodels[13,14,15]. Since it is well understood, this TERSM problem provided an excellent baseline for our feasibility studies.

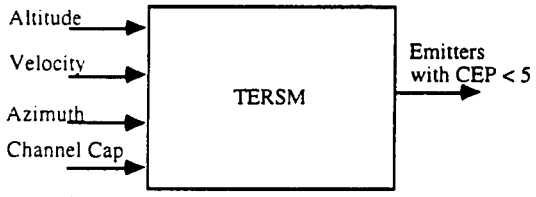


Figure 4.1. Baseline TERSM problem.

Table 4.1. Range of interest for each input variable.

Input Variable	Lower Limit	Upper Limit
Altitude (feet)	5000	40,000
Velocity (knots)	186	1150
Azimuth (degrees)	60	150
Channel Capacity	4	30

Using a layered central composite design, Caughlin[13], compiled a table of 49 input/output pairs from which he obtained the following reduced 4*th* order polynomial metamodel,

$$\sqrt{y} = 22.4331 - 0.0148x_1 - 2.7822x_2 + 0.1432x_3 + 3.1432x_4 + 0.3653x_1x_3 + 1.2439x_1x_4 + 0.1483x_2x_3 + 0.4430x_2x_4$$
$$+0.2698x_3x_4 + 0.4369x_1x_2x_3 + 0.3286x_1x_2x_4 + 0.0960x_1x_3x_4 - 0.2791x_2x_3x_4 - 0.8326x_1^2 + 0.7642x_3^2 - 1.8413x_4^2$$
$$+0.7577x_1^3 + 4.9038x_2^3 + 1.0924x_3^3 - 1.1907x_1^4 - 4.8443x_2^4$$

In the equation above, the input x_1 is the altitude, x_2 is the velocity, x_3 is the azimuth angle, and x_4 is the channel capacity. All inputs are centered and scaled (so that the upper limit maps to 1, the lower limit to –1, and the middle value to 0). Using the TERSM software, we were able to duplicate Caughlin's results. This served to validate our version of the TERSM simulation software.

To obtain a CCNN metamodel, we randomly generated 238 input/output pairs using TERSM, and used them as a training set. At the end of training the CCNN had grown to only 19 neurons in the hidden layer. To compare the polynomial and CCNN metamodels, we used TERSM to obtain an independent testing set consisting of 165 randomly selected input/output pairs. Table 4.2 below summarizes the results.

Table 4.2. MSE comparison between the polynomial and CCNN for baseline TERSM.

	49 point set	238 point NN set	165 point test set
polynomial	518.40	575.86	795.20
CCNN	497.75	353.25	417.50

In Table 4.2, the 49 point set is the set of input/output pairs that were used to obtain the polynomial coefficients. This is the *training set for the polynomial*. The 238 point set is the set of input/output data pairs that were used to train the CCNN. This is the *training set for the CCNN*. The 165 point set is independent of the other two (i.e., has no points in common with either of them) and represents an independent *testing set*. The purpose of the testing set is to see how well the metamodels generalize to data which they were not exposed to during training. Generalization involves both *extrapolation* to parts of the input space which are outside the range seen during training, and it involves *interpolation* to parts of the input space which lie "between" points seen during training. As expected, the polynomial does well on its training set, and the CCNN does well on its training set also. The CCNN, however, does much better than the polynomial on the testing set. Since a metamodel is primarily going to be used to generalize, good performance on the testing set is the most important performance criterion.

4.2. A "Tougher" TERSM Problem.

For the baseline TERSM problem, the number of emitters detected changes in a more or less "continuous" way with changes in the input variables, leading to a relatively "smooth" response surface. For example, increasing the altitude allows the sensor to see further over the horizon and detect more emitters. Increasing the velocity results in less emitters being detected because the aircraft is over the emitter field for a shorter period of time. The azimuth was found to have little effect, and increasing the channel capacity results in a marginal increase in the number of emitters detected. Thus, the baseline TERSM problem has a response surface which is easy to fit with a low order polynomial. The relative smoothness of the response surface in agreement with the fact that the CCNN was able to give such good performance with only 19 neurons in its hidden layer.

This prompted us to develop a "tougher" TERSM problem with a "rougher" and more challenging response surface. For the tougher TERSM problem we looked at the functional relationship between the four inputs and single output shown in Fig. 4.2 for the range of inputs in Table 4.3. As for the baseline TERSM problem, the output is the number of emitters detected which have a CEP less than 5 nautical miles. In Fig. 4.2 the four inputs are: the initial aircraft coordinates Xo and Yo, the initial aircraft compass heading θ (the aircraft flies a constant heading), and the aircraft velocity V (the aircraft flies at a constant velocity).

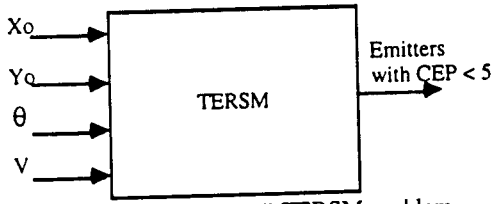


Figure 4.2. "Tougher" TERSM problem.

Table 4.3. Range of interest for each input variable.

Input Variable	Lower Limit	Upper Limit
Xo	0	400
Yo	0	300
Heading (degrees)	0	359
Velocity (knots)	186	1150

To get some insight as to why this TERSM problem is "tougher" consider the following. The TERSM experiments were conducted using an emitter set which has 2359 emitters all located in a rectangular region on the surface of a sphere. Imagine now a set of experiments where the initial location of the aircraft is always to the north of this rectangular region, and suppose that for each experiment we only change the aircraft heading (the aircraft flies a constant heading for the duration of its mission). For a zero heading angle (due north) we are flying away from the emitter field and no emitters are detected. As the angle increases towards 180 degrees (due south) more and more emitters are detected as we fly over an increasingly larger portion of the emitter field. Then as the angle increases beyond 180 degrees, the number of emitters detected decreases. Thus, the tougher TERSM problem has a response surface consisting of a collection of isolated "humps" where many emitters are detected, surrounded by large flat regions where no emitters are detected. Such a response surface should be quite difficult for a low order polynomial to capture. The CCNN, however, due to its universal function approximation capabilities, should be able to approximate the response surface. Nevertheless, such a response surface still poses difficulties for the CCNN in terms of data collection. If the enough points are not chosen, it is possible to miss some the humps entirely.

As before, we performed simulation experiments to construct three data sets: a training set for the polynomial, a randomly chosen training set for the CCNN, and a randomly chosen testing set with which to compare the two. The polynomial training set was obtained using a layered CCD design consisting of the 49 input/output pairs. The CCNN training set consisted of 14,641 input/output pairs, and the independent testing set consisted of 500 input/output pairs.

Using the 49 point set we obtained the following 4th order polynomial metamodel. As for the baseline TERSM problem, a square root nonlinearity on the output gives the best results:

$$\sqrt{y} = 20.3885 - 4.8499x_1 + 7.7436x_2 - 2.6979x_3 - 2.9343x_4 - 1.2951x_1x_2 + 3.6486x_1x_3 + 1.2144x_1x_4 + 1.9655x_2x_3 - 2.3306x_2x_4$$
$$+0.0265x_3x_4 - 1.0556x_1x_2x_3 - 1.0158x_1x_2x_4 + 1.4222x_1x_3x_4 + 0.2683x_2x_3x_4 - 1.0927x_1x_2x_3x_4 - 13.5657x_1^2 - 12.1073x_2^2$$
$$+0.6690x_3^2 + 3.1144x_4^2 + 6.4287x_1^3 - 9.1515x_2^3 + 3.3922x_3^3 + 5.2131x_4^3 + 4.7740x_1^4 + 4.6920x_2^4 - 0.6565x_3^4 - 2.9759x_4^4$$

In the equation above, x_1 is the initial aircraft x-location, x_2 is the initial aircraft y-location, x_3 is the aircraft heading (0 degrees is due north), and x_4 is the aircraft velocity. All inputs are centered and scaled, and the output y is the number of emitters located with a CEP less than 5 nautical miles.

Comparative results are shown in Table 4.4. The ruggedness of the response surface resulted in the polynomial giving very poor performance. Even on the 49 point set, the one used to obtain the polynomial coefficients, the MSE was 37,278. That is, on the average, the output of the polynomial and the output of TERSM differed by 193 emitters, a huge difference when one considers that the number of emitters detected with a CEP < 5 nautical miles averaged only 243 and never exceeded 849. The CCNN gives much better performance following a rigorous training process consisting of many more training points.

Table 4.4. MSE comparison between the polynomial and CCNN for "tougher" TERSM.

	49 point set	14,641 point NN set	500 point test set
polynomial	37,278	66,035	61,638
CCNN	5,243	4,379	4,671

4.3. Baseline ARMS problem.

Next we present metamodeling results using the ARMS model from Fig. 4.3 for the range of inputs in Table 4.5. The inputs λ_i are the arrival rates for the three customer classes, θ is the number of tokens, and the output, $J = s_1$, is the service time of the class 1 (highest priority) customers. The service time is the time from the customer arrival at the system to the time the customer finishes service and departs the system. In the context of aircraft refueling and maintenance, this is the "downtime" or amount of time that an aircraft is not available for service.

From the preliminary analysis in Cassandras and Gong[6] we already have some intuition about how we can expect ARMS to behave. When the number of tokens θ is small, the system will appear to the class 1 customers as if they are the only ones using the system. This is because, regardless of the arrival rates of the lower priority customers, if there are customers waiting in the class 1 arrival queue, then those customers will get the tokens as they become available. When there are not many tokens, the queue at the server will be short, and the customer will pass quickly through the system. As the arrival rate of the class 1 customers, λ_1, increases towards the service capability of the server, $1/\mu$, arriving class 1 customers will tend to find long queues in the arrival queue, their

waiting time in the arrival queue will be long, and thus their service times will increase. It is only when the arrival rate of the class 1 customers is relatively low, the number of tokens is large, and the arrival rate of the other classes of customers is high that the class 1 customers will feel the effects of the lower priority customers. This is because when the arrival rate of the class 1 customers is low, the class 1 arrival queue will often be empty, and lower class customers will be able to get tokens more often. As a result, when a class 1 customer arrives and gets a token, it will find many other customers ahead of it in the server queue. Thus, although the waiting time in the arrival queue will be low, the waiting time in the server queue will be longer because there is no priority there. Thus we get a response surface that increases asymptotically with λ_1 and monotonically with θ.

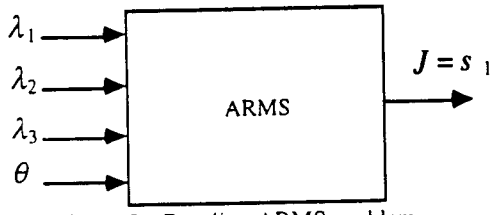


Fig. 4.3. Baseline ARMS problem.

Table 4.5. Range of interest for each input variable.

Input Variable	Lower Limit	Upper Limit
λ_1 (customers/sec)	0.1	1.0
λ_2	0.1	1.0
λ_3	0.1	1.0
θ	1	20

Following our usual approach, we used a layered CCD design to collect 49 training points for a polynomial metamodel, 500 randomly selected points to train a CCNN metamodel, and 500 randomly selected points for comparing the two models. The 49 points gave the following 4*th* order polynomial metamodel. As with all previous problems, a square root transformation on the output gives the best MSE performance:

$$\sqrt{y} = 3.6325 - 0.1023x_1 - 0.1238x_2 - 0.0383x_3 + 1.3530x_4 - 0.0129x_1x_2 - 0.1893x_1x_3 + 0.1437x_1x_4 - 0.5784x_2x_3 + 0.2346x_2x_4$$
$$+0.2536x_3x_4 - 0.1399x_1x_2x_3 - 0.1262x_1x_2x_4 - 0.0960x_1x_3x_4 - 0.4492x_2x_3x_4 - 0.0848x_1x_2x_3x_4 - 0.4117x_1^2 + 0.1263x_2^2$$
$$-0.4416x_3^2 - 0.0308x_4^2 + 1.7967x_1^3 + 0.5235x_2^3 + 0.2576x_3^3 + 0.0052x_4^3 + 1.9208x_1^4 - 0.2488x_2^4 + 0.1862x_3^4 - 0.4665x_4^4$$

Table 4.6 summarizes our comparative results for the baseline ARMS problem.

Table 4.6. Comparison between the polynomial and CCNN for baseline ARMS.

	49 point set	500 point NN set	500 point test set
polynomial	1.87	22.76	17.74 (34.3%)
CCNN I	85.81	3.44	13.01 (22.9%)
CCNN II	86.69	6.89	10.57 (16.3%)

In Table 4.6, CCNN I was trained to learn a scaled version of the output, $0.1J$, and CCNN II was trained to learn the natural log of the output, $\log(J)$. Because of the asymptotic behavior of ARMS, it was felt that the log would "flatten" the output curve making the learning task easier. Interestingly, the performance of the two networks is almost identical as was the final number of hidden layer neurons in each: 25 for CCNN I and 23 for CCNN II. In comparing the CCNN to the polynomial metamodel, performance on the 500 point test set is really the most important since it provides a measure of the generalization capability over a representative sample of the input space (unlike the 49 point set which covers the extreme cases). In terms of the MSE criterion, both the CCNN and the polynomial appear to generalize well. When the range of the output is small, as it is in this case, however, MSE results can be misleading. A better performance measure to use in such cases is the Mean Squared Relative Error (MSRE), given by,

$$MSRE = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{e_i}{y_i}\right)^2 \times 100$$

where e_i is the error between the target and network output and y_i is the target output. This criterion (shown in parenthesis in Table 4.6) clearly demonstrates the superiority of the CCNN with the log transformation.

4.4. Concurrent simulation to speed data collection in the ARMS model

For DES like the ARMS model, concurrent estimation can be used to speed the process of collecting the input/output pairs needed to obtain a metamodel. One of the parameters that affects the system time of the class 1 customers in the ARMS model is the number of tokens θ. As described previously, changing the number of tokens changes their resulting system times, with a smaller number of tokens favoring the higher priority customers (see Fig. 4.4).

Since analytical performance measures for this system are hard to derive, in order to determine the function in Fig. 4.4 it is necessary to resort to simulation with at least one simulation for each different value of θ_i. As described in Section 2.3, concurrent estimation techniques allow one the obtain performance estimates under any number of tokens $\theta_1 \ldots \theta_N$ by observing *only a single* sample path under some θ_0 tokens. Generally, the amount of time to collect these $N+1$ performance estimates will be much less than the time required to run $N+1$ individual "brute force" simulation experiments.

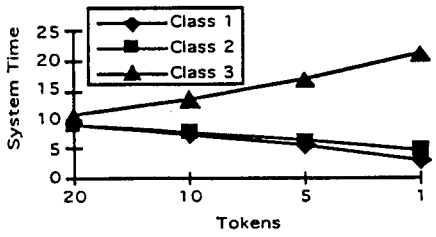


Figure 4.4. Effect of changing the number of tokens (θ)on the system time.

To illustrate just how significant this speedup can be, consider the ARMS model from Fig. 3.3. Assume that all arrival queues have infinite capacity, all arrival processes are Poisson, and all customer classes have identical (exponentially distributed) service time requirements. Under these conditions, Fig. 4.5 shows the speedup that can be achieved using some recent extensions to the TWA scheme mentioned in Section 2.3. In the figure, speedup_1 corresponds to the average speedup achieved over *all* concurrently constructed sample paths. Speedup_2 corresponds to the marginal speedup when going from n to $n+1$ parallel sample paths.

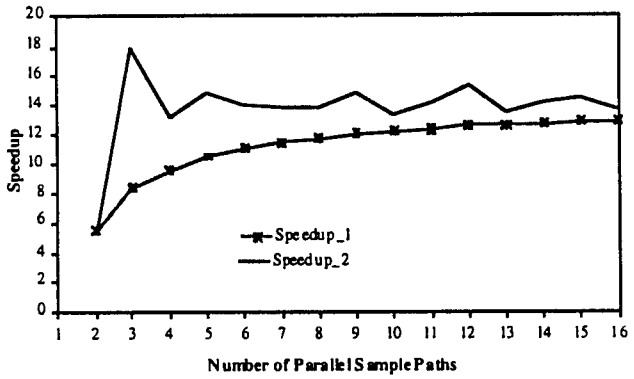


Figure 4.5. Speedup factor for concurrent estimation.

5. CONCLUSIONS

Simulation is replacing actual experiment for reasons of cost, safety, and practicality. One application area where simulation is seeing an increased use is in design and decision making for C^3I systems. Such issues typically require simulation to answer a multitude of "what if" questions. The difficulty is that complex simulations can be very time consuming even on modern supercomputers. What is needed, therefore, is a way to speed the processes of evaluating the outcome of competing "what if" scenarios. This is the motivation behind the idea of metamodeling—building a simpler higher-level model of a simulation model. This is a function approximation task, i.e., replacing the simulation with some function which has the same input/output behavior as the simulator. The benefit of such an approach is that evaluating the function is orders of magnitude faster than running the simulation, allowing for real-time decision making. Traditionally, polynomials have been used for metamodeling. In this paper, we proposed neural networks as an alternative metamodeling device in order to exploit their "universal function" approximation capability and their superior scalability to complex function approximation problems which have a large number of inputs and outputs. Polynomials, on the other hand, have known limitations in terms of their function approximation power, and they do not scale well as the problem size and complexity increases. In comparing polynomials to neural networks, we saw that constructing polynomial metamodels requires a great deal of technical sophistication on the part of the metamodeling practitioner. First, one has construct an experimental design to decide what points to collect for obtaining the coefficients of the polynomial. Second, one must examine the residual errors to decide when certain terms are not significant, in which case their coefficients are set to zero. Third, for many polynomial curve fitting problems, it is advantageous to use a nonlinear transformation, such as the square root, on the output. In contrast, constructing

metamodels using neural networks is easier as it simply involves randomly selecting a set of training points and presenting them to a neural network training algorithm. Our numerical performance results demonstrate the superiority of the CCNN over polynomials in terms of generalization capability on several nontrivial metamodeling examples.

Neural networks, however, are not a panacea as they also involve some challenging issues. The first issue involves the determination of the network size (i.e., the number of neurons to use in the hidden layer). Too few and the neural network will underfit, too many and the neural network will overfit. To deal with this issue we chose the Cascade Correlation Neural Network (CCNN) architecture because it automatically decides how many hidden neurons are necessary during learning. Second, neural network training algorithms can be very slow to converge. The CCNN also addresses this difficulty, and uses a training algorithm that is generally faster than methods such as the usual error backproprogation algorithm. A third difficulty is the number of input/output pairs needed to train the neural network. For DES simulations we demonstrated concurrent estimation as a way to collect several input/output pairs from a single simulation run in much less time than it would take to collect these same points via individual simulation experiments. In addition to concurrent estimation, what is needed is some "adaptive" data collection strategy which can use the history of the points already sampled to decide where to sample next. The objective of such a strategy is to locate the "bumps" in the output function so as to take few samples where the function is "flat" and more samples from the "bumps." One idea here is to use perturbation analysis techniques for DES[18,19] to extract derivative information from each simulation run. One can then use these derivative estimates as local "roughness" measures to decide where to sample next. This sort of "adaptive intelligent experimental design" remains an open issue and a subject of our ongoing research.

ACKNOWLEDGMENT

This work is supported by the Air Force Rome Laboratory under Contracts F30602-94-0109, F30602-95-C-0242 F30602-97-C-0125.

REFERENCES

1. Astrom, K.J. and B. Wittenmark, *Adaptive Control* (2ed.), Addison Wesley, 1995.
2. Montgomery, D.C., *Design and Analysis of Experiments*, John-Wiley, 1984.
3. Hornik, K., M. Stinchcombe, and H. White, "Multilayer Feedforward Networks are Universal Approximators," *Neural Networks*, Vol. 2, pp. 359-366. 1989.
4. Fausett, L., *Fundamentals of Neural Networks, Architecture, Algorithms and Applications*, Prentice Hall, 1994.
5. Fahlman, S.E., and Lebiere, C., "The Cascade-Correlation Learning Architecture", Carnegie Mellon Univ., Technical Report CMU-CS-90-100, Feb., 1990.
6. Cassandras, C.G. and W-B. Gong, "Enabling Technologies for Real-Time Simulation," *Rome Lab. IRAE Technical Rep.*, 1996.
7. Cassandras, C.G. and S.G. Strickland, "Observable Augmented Systems for Sensitivity Analysis of Markov and Semi-Markov Processes," *IEEE Trans. on Automatic Control*, Vol. AC-34(10): 1026-1037, 1989.
8. Vakili, P., "A Standard Clock Technique for Efficient Simulation," *Operations Research Letters*, Vol. 10, pp. 445-452, 1991.
9. Cassandras, C.G. and S.G. Strickland, "On-Line Sensitivity Analysis of Markov Chains," *IEEE Trans. on Automatic Control*, Vol. AC-34(1): 76-86, 1989.
10. Cassandras, C.G. and J. Pan, "Parallel Sample Path Generation for Discrete Event Systems and the Traffic Smoothing Problem," *Journal of Discrete Event Dynamic Systems*, Vol. 5, No. 2/3, pp. 187-217, 1995.
11. Cassandras, C.G. and C. Panayiotou, "Concurrent Sample Path Estimation for Discrete Event Systems," *Proceedings of 35th IEEE Conference on Decision and Control*, pp. 3332-3337, 1996 (also subm. to *Journal of Discrete Event Dynamic Systems*).
12. Tatum, F.A., "A Tactical Electronic Reconnaissance Simulation Model," *Rand Corporation Technical Report*, 24 October 1969.
13. Caughlin, D., "Verification, Validation, and Accreditation of Models and Simulations through Reduced Order Metamodels," in *Proceedings of the 1995 Winter Simulation Conference*, pp. 1405-1412, December 1995.
14. Zeimer, M.A., and J.D. Tew, "Metamodel Applications Using TERSM," in *Proceedings of the 1995 Winter Simulation Conference*, pp. 1421-1428, December 1995.
15. Zeimer, M.A., J.D. Tew, R.G. Sargent, and A. Sisti, "Metamodel Procedures for Air Engagement Simulation Models," *IRAE Technical Report*, Rome Laboratory, Griffis A.F.B., N.Y., January 1993.
16. Meidt, G. and Bauer, K.W. Jr., "PCRSM: A Decision Support System for Simulation Metamodel Construction", *Simulation*, Vol. 59, 3, pp. 183-191, 1992.
17. Percich, D.M., "Modeling the Permanent Change of Station Moving Costs of Strategic Airlift Pilots", MS Thesis, Air Force Institute of Technology, 1987.
18. Glasserman, P., *Gradient Estimation Via Perturbation Analysis*. Boston, MA: Kluwer, 1991.
19. Ho, Y.C., and X.R. Cao, *Perturbation Analysis of Discrete Event Dynamic Systems*. Kluwer Publishing, 1991.

eU.S. GOVERNMENT PRINTING OFFICE: 1999-510-079-81215