Performance and Reliability Analysis Using Directed Acyclic Graphs

ROBIN A. SAHNER AND KISHOR S. TRIVEDI, MEMBER, IEEE

Abstract—A graph-based modeling technique has been developed for the stochastic analysis of systems containing concurrency. The basis of the technique is the use of directed acyclic graphs. These graphs represent event-precedence networks where activities may occur serially, probabilistically, or concurrently. When a set of activities occurs concurrently, the condition for the set of activities to complete is that a specified number of the activities must complete. This includes the special cases that one or all of the activities must complete. The cumulative distribution function associated with an activity is assumed to have exponential polynomial form. Further generality is obtained by allowing these distributions to have a mass at the origin and/or at infinity. The distribution function for the time taken to complete the entire graph is computed symbolically in the time parameter t. The technique allows two or more graphs to be combined hierarchically. Applications of the technique to the evaluation of concurrent program execution time and to the reliability analysis of fault-tolerant systems are discussed.

Index Terms—Availability, directed acyclic graphs, fault-tolerance, Markov models, performance evaluation, program performance, reliability.

I. Introduction

GRAPH-BASED models are often used to analyze computer systems. For example, precedence graphs are used to analyze the performance of concurrent programs [2], [18], fault-trees and reliability block diagrams are used to analyze system reliability and availability [21], and Markov and semi-Markov models are used for analyzing both performance and reliability/availability [25]. This paper describes a graph-based modeling technique that is efficient and can solve a wide class of problems. The technique includes the use of directed acyclic graphs, fault-trees, reliability block diagrams, and acyclic Markov and semi-Markov chains. It has been implemented as a software tool. The technique is illustrated by means of several examples.

Graph models are commonly used to study the behavior of systems that contain concurrency. The granularity of activities varies from model to model. An activity may represent a single machine instruction or it may represent an entire process. Activities may have constant or random duration. The degree of concurrency may be limited or unlimited. A particular model may assign activities to nodes and use edges for precedence [12], or it may assign activities to edges and use the nodes for precedence specification [13]. Many models assume that all activities must occur [2]. Some allow probabilistic branching [27].

When the activity times are constant and there are infinite resources, it is easy to compute the completion time for the entire set of activities composing a graph. When activity times are allowed to be random, the analysis of a graph model is extremely difficult. The source of the difficulty is the fact that if two paths in the graph have one or more edges in common, the random variables for the finishing time of the paths are not independent.

There is a long history of investigation of this abstract problem in the context of PERT (Program Evaluation Review Technique) networks [5]. The networks may be analyzed for mean completion time, for time of shortest route, and for various other measures, such as the probability that a path is the shortest path. Because of the dependencies among paths, the exact analysis of a network involves investigating each path assuming a constant time for the common edges, then combining the individual results using integral formulas containing conditional distributions [9], [16]. In [22], an approach has been developed that conditions on the edges in a uniformly directed cut set and then evaluates the integral formulas numerically using Monte-Carlo simulation. The exact solution is time-consuming, so work has been done in obtaining efficient approximations and bounds [5], [8].

If all activity times are memoryless, an alternative to the use of conditioning is to transform the graph into a continuous-time Markov chain [11], [25]. The chain can then be analyzed by using one of a number of numerical methods. The main drawback of this alternative is that the number of states in the Markov chain increases exponentially with the number of nodes in the graph. The modeler must also contend with the practical difficulties of generating the states and transitions of the Markov chain and solving a very large system of ordinary differential equations.

If the structure of the graphs is restricted, the problem becomes less complex. Many problems that are NP-complete for arbitrary graphs can be solved in linear or polynomial time for series-parallel graphs, because of their restricted structure. Such problems include many graph-

Manuscript received April 30, 1985; revised February 28, 1986. This work was supported in part by the Air Force Office of Scientific Research under Grant AFOSR-84-0132, by the Army Research Office under Contract DAAG29-84-0045, and by the National Science Foundation under Grant MCS-830200.

R. A. Sahner is with Gould Computer Systems Division, Urbana, IL 61801.

K. S. Trivedi is with the Department of Computer Science, Duke University, Durham, NC 27706.

IEEE Log Number 8716568.

theoretical problems [23] and also problems such as scheduling, for which a graph is the underlying model [12]. If an event-precedence graph with random activity times is series-parallel, the distribution function of the completion time of the graph can be obtained exactly in linear time by combining the distribution functions of the individual nodes using multiplication and convolution.

Martin [13] applied this approach to the analysis of PERT networks. The distributions were assumed to be piecewise polynomials defined over a finite range. More recently, Robinson [18] and Kleinoder [10] have also used series-parallel graphs to analyze the performance of programs that contain concurrent tasks. When tasks or subgraphs are executed in parallel, all of them must finish before any successor tasks can be started. The graphs are analyzed by performing numerical calculations on distribution functions obtained experimentally; Kleinoder also allows phase-type distributions, which are approximated and represented internally as sums of discrete density functions.

Such series-parallel models are a special case of our modeling technique, which is called SPADE (*S*eries-*PA*rallel *D*irected acyclic graph *E*valuator). We have generalized the series-parallel graph model in a way that allows it to be used not only for performance analysis, but also for reliability and (under certain restrictions) availability analysis. The key features of the technique are:

- Each graph node is assigned a function that is allowed to be any exponential polynomial that is real-valued and has range between zero and one. The functions are allowed to have a mass at zero and/or at infinity. This class of functions includes all of the Coxian phase-type distributions [3].
- No assumptions are made about the nature of the "activities" associated with the nodes. It is this fact that allows our technique to be used for performance, reliability and availability analysis. When used for performance analysis, the function $F(t)$ associated with a node represents the probability that a task finishes by time t. When used for reliability analysis, $F(t)$ represents the probability that a component has failed by time t. When used for availability analysis, $F(t)$ denotes the probability that a component is not available at time t.
- The interpretation of parallel subgraphs is chosen from a general and useful group of alternatives. Parallel subgraphs are not required to have identical distributions.
- The analysis of a graph is done symbolically in t, and is exact (up to truncation error).
- Because results are symbolic, it is possible to use the solution of a graph as the function assigned to a node in another graph. Thus it is possible to combine graphs hierarchically. The facility for using a hierarchy of graphs is built into the modeling technique, but the choice of how to arrange the hierarchy is left to the modeler.
- The technique is suitable for efficient automation, and has been automated in the form of a software tool, also called SPADE.
- Although the model is described here in the form of a series-parallel graph, it includes as special cases reliability block diagrams [7], [14], fault trees without repeated components [21], and acyclic Markov and semi-Markov chains [25]. Each of these models can be transformed in linear time to a series-parallel graph model. For details on how this can be done, see [19]. The SPADE program accepts the specification of the specialized models, in addition to the general series-parallel graph model.

In the next section, we describe the SPADE modeling technique. In Section III we briefly describe the SPADE program. Section IV gives examples illustrating the use of our approach. Although the examples are simple, more complex problems can be and have been analyzed using our modeling technique.

II. The SPADE Modeling Technique

The SPADE modeling technique is a method for analyzing systems whose underlying structure can be modeled as a series-parallel graph or a hierarchy of such graphs. Graph nodes correspond to components of a system or subsystem. Each graph node is assigned a function of one variable t, generally representing time. Each set of parallel subgraphs is assigned one of several alternative types. Analysis of the graph results in a function in t. The interpretation of the function produced by the analysis depends on the interpretation of the functions assigned to the individual nodes.

To use SPADE for performance analysis, each node would be assigned the cumulative distribution function (CDF) for the completion time of the task it models. Then the result function is the CDF for the completion time of the graph consisting of these tasks. To use SPADE for reliability analysis, each node would be assigned the CDF for the time-to-failure of a component. Then the result function is the CDF for the time-to-failure for the system modeled by the graph. To use SPADE for availability analysis, each node would be assigned a function that gives the probability that a component is unavailable at a given time. Then the result function gives the probability that the system as a whole is unavailable at a given time.

This section presents a definition of "series-parallel" graphs and defines how a CDF is obtained for a graph based on the CDFs of the individual nodes.

A. Series-Parallel Graphs

There are many definitions for the term "series-parallel," and many different names for the series-parallel structure. In [10], such graphs are called "simple," in [11] they are called "standard," in [9] they are called "uncrossed," and in [5] they are called "completely reducible." Informally, series-parallel graphs can be described as being "well-structured." They are built by starting with single nodes. These single nodes may be combined, either in series or in parallel. The subgraphs obtained in that way may in turn be combined, in series or in parallel, with other nodes or subgraphs. An important characteristic of series-parallel graphs is that whenever multiple edges leave a node, they lead to two or more *disjoint* parallel subgraphs. Once an activity with multiple successor activities is finished, the disjoint subgraphs that follow it proceed in parallel, with neither having any dependencies on what is happening in the other subgraph.

To define "series-parallel" more formally, we begin by defining a finite linear directed graph to be an ordered quadruple $G = (N, A, S, T)$ where

a) N is a finite set of elements called nodes.

b) A is a subset of $N \times N$, called the set of edges.

c) S is the subset of N containing those nodes that have no incoming edges. These are the entrance nodes.

d) T is the subset of N containing those nodes that have no outgoing edges. These are the exit nodes.

Suppose $G_1 = (N_1, A_1, S_1, T_1)$ and $G_2 = (N_2, A_2, S_2, T_2)$ are nonintersecting graphs. A graph $G = (N, A, S, T)$ is the series connection of G_1 and G_2 if and only if

a) At least one of $|T_1|$ and $|S_2|$ is 1. That is, at least one of the two sets is a singleton.

b) $N = N_1 \cup N_2$.

c) $A = A_1 \cup A_2 \cup (T_1 \times S_2)$.

d) $S = S_1$, $T = T_2$.

A graph G is the parallel combination of G_1 and G_2 if and only if

a) $N = N_1 \cup N_2$.

b) $A = A_1 \cup A_2$.

c) $S = S_1 \cup S_2$, $T = T_1 \cup T_2$.

The class of series-parallel graphs is the smallest class of graphs containing the unit graphs (graphs consisting of one node) and having the property that whenever G is the series or parallel connection of two graphs in the class, then G is in the class.

Note that a series-parallel graph is by definition acyclic. Our definition of series-parallel also implies that a series-parallel graph cannot have redundant edges. That is, if there is an edge from node x to node y, there is no other path from x to y.

Fig. 1 shows several examples of series-parallel graphs. In this and all subsequent figures, the direction of the edges is not shown explicitly; it is assumed that an edge points downward. In Fig. 2, graph G1 is "transitive series-parallel" [26] but is not series-parallel because of the redundant arc from A to C. It can be transformed into the series-parallel graph G2 by adding the dummy node Z, whose activity takes no time. Graph G3 in Fig. 2 is not series-parallel under any definition.

Although the definition of series-parallel is binary in nature, it is convenient to think of each series or parallel combination as being built from n subgraphs, rather than two. The parallel (series) combination of n subgraphs is defined to be the sequence of $n - 1$ binary parallel (series) combinations of the subgraphs.

B. Distribution Function of a Graph

With each node in a graph is associated a real-valued function whose range is $[0, 1]$. The class of allowed functions will be described in Section II-E. It is also possible

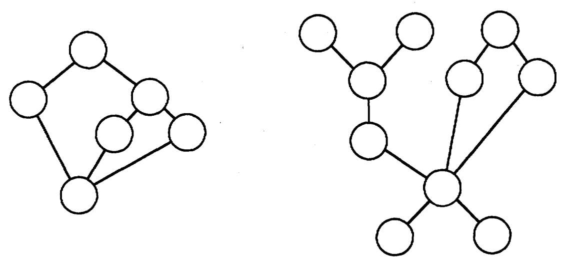


Fig. 1. Examples of series-parallel graphs.

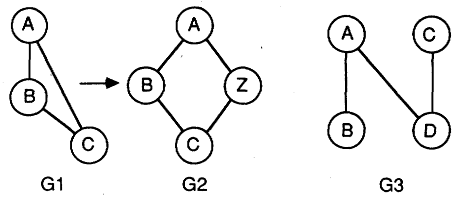


Fig. 2. Graph examples.

to assign to a node a function obtained by analyzing some graph.

Each set of parallel subgraphs is specified to be either probabilistic or concurrent. If the set is probabilisitic, each subgraph is assigned a probability value. If the set is concurrent, the set of n subgraphs is assigned a type and an integer $k \leq n$.

We define the cumulative distribution function (CDF) of a graph recursively. In this section, we simply define how the function is computed. In the next section, we will explain how the computation of the function can be interpreted in terms of performance, reliability, and availability analysis.

If G consists of a single node, the distribution function is given as part of the model specification.

Suppose G was formed by combining the graphs G_1, $G_2, \cdots, G_n$ having independent distribution functions $F_1, F_2, \cdots, F_n$. If the graphs were combined in series, the distribution function for the graph G is defined by

$$F_{\text{sum}}(t) = \bigotimes_{i=1}^{n} F_i(t) \tag{1}$$

where the symbol $\otimes$ represents convolution. The convolution of two CDF's F_j and F_k is defined by

$$F_j(t) \otimes F_k(t) = \int_0^t F_k(t - x)\, dF_j(x).$$

Note that the order of the subgraphs does not matter.

If the subgraphs G_i were combined in parallel, the distribution function of the graph depends on whether the subgraphs are probabilistic or concurrent. For probabilistic parallel subgraphs, suppose the probability assigned to G_i is p_i. Then the distribution function of G is given by

$$F_{\text{prob}}(t) = \sum_{i=1}^{n} p_i F_i(t). \tag{2}$$

For concurrent parallel subgraphs, the distribution function for G is given by the kth order statistic [4] of the n subgraph functions. Two special cases occur so often that we treat them separately. If $k = 1$ then the distribution is the "minimum" of the distributions of the subgraphs. In that case we have

$$F_{\min}(t) = 1 - \prod_{i=1}^{n} (1 - F_i(t)). \tag{3}$$

If $k = n$ then the distribution is the "maximum" of those of the subgraphs, and we have

$$F_{\max}(t) = \prod_{i=1}^{n} F_i(t). \tag{4}$$

In the general case, if the n subgraphs have identical distributions F then the distribution for G is (omitting the parameter t for better readability)

$$F_{k/n} = \sum_{i=k}^{n} \binom{n}{i} F^i (1 - F)^{n-i}. \tag{5}$$

If the subgraphs do not have identical distributions, the expression for $F_{k/n}$ is (see [4])

$$F_{k/n} = \sum_{|T| \ge k} \left(\prod_{j \in T} F_j \right) \left(\prod_{j \notin T} (1 - F_j) \right) \tag{6}$$

where T is a set of indexes ranging over all combinations of k or more indexes chosen from $\{1, 2, \cdots, n\}$. That is, T ranges over all choices $\{j_1, j_2, \cdots, j_m\}$ such that $k \le m \le n$ and $j_1 < j_2 < \cdots < j_m$.

Equation (6) can be written in a form more suitable for mechanical computation. Given a vector $\vec{F} = (F_1, F_2, \cdots, F_n)$, let $S_i(\vec{F})$ be the elementary symmetric polynomial of degree i in $\vec{F}$. That is,

$$S_i(\vec{F}) = \sum_{|U| = i} \prod_{j \in U} F_j$$

where U ranges over all combinations of i indexes chosen from $\{1, 2, \cdots, n\}$. Then we have the following lemma.

Lemma: If n subgraphs have distributions given by the elements of the vector $\vec{F}$, the distribution for the kth subgraph to finish is given by

$$F_{k/n}(\vec{F}) = \sum_{i=k}^{n} (-1)^{i-k} \binom{i-1}{k-1} S_i(\vec{F}). \tag{6a}$$

Proof: First we expand the second product in (6), so that we have

$$F_{k/n}(\vec{F}) = \sum_{|T| \ge k} \left(\prod_{j \in T} F_j \right) \left(\sum_{V \subseteq T^c} (-1)^{|V|} \prod_{j \in V} F_j \right)$$
$$= \sum_{\substack{|T| \ge k, \\ V \subseteq T^c}} \left((-1)^{|V|} \prod_{j \in T \cup V} F_j \right).$$

Now we are interested in obtaining the coefficient for a particular product $F_{j_1} F_{j_2} \cdots F_{j_i}$ where $i \ge k$. That product appears whenever $T \subseteq \{j_1, \cdots, j_i\}$ and $|T| \ge k$. For each $m \ge k$, there are $\binom{i}{m}$ ways of choosing T. Once T is chosen, V is determined and $|V| = i - m$. Therefore the coefficient of interest is (see [17])

$$\sum_{m=k}^{i} (-1)^{i-m} \binom{i}{m}$$
$$= \sum_{m=0}^{i-k} (-1)^m \binom{i}{m}$$
$$= \sum_{m=0}^{i-k} (-1)^m \left[\binom{i-1}{m} + \binom{i-1}{m-1} \right]$$
$$= (-1)^{i-m} \binom{i-1}{k-1}.$$

Noting that this coefficient is independent of the particular i-tuple chosen, and that the sum of all such i-tuples is exactly the elementary symmetric polynomial $S_i(\vec{F})$, we have the desired result.

Formula (6a) is computationally much more tractable than (6), because the elementary symmetric polynomials can be computed efficiently. Let us write $S_i(j)$ for the elementary symmetric polynomial of degree i chosen out of a vector $\vec{F}$ with j components. We can compute the polynomials $S_i(j)$ as follows:

a) $S_1(1) = F_1$.
b) $S_1(j) = S_1(j-1) + F_j$ for $j > 1$.
c) $S_j(j) = S_{j-1}(j-1) F_j$ for $j > 1$.
d) $S_i(j) = S_i(j-1) + (F_j S_{i-1}(j-1))$ for $1 < i < j$.

Note that we do not need to compute all of the terms $S_i(j)$, but only the last $n - k + 1$ terms for each j. Using this method, the number of multiplications of CDF's is $O(n(n-k))$.

C. Graph Interpretation

Suppose we are modeling program execution. Clearly, series subgraphs represent serial statement execution. The interpretation of parallel subgraphs depends on the type of parallelism. We assign the type maximum to a parallel combination of subgraphs in order to model the concurrent execution of groups of tasks, with each group running to completion. The set of groups is not considered finished until all task groups are finished. This construct models the statement types *cobegin* or *parbegin* in various concurrent programming languages. This is the only kind of concurrency considered in PERT networks and by Robinson [18] and Kleinoder [10]. If parallel subgraphs are probabilistic, they can be interpreted as the alternatives in a conditional statement (*if* or *case* statement types in programming languages).

Parallel subgraphs of the type minimum can model the parallel execution of a nondeterministic algorithm in which the verification of all guessed solutions is attempted concurrently, and the first guess to be verified provides a solution to the whole problem. Another example of the use of parallel subgraphs of type minimum is in modeling the parallel search of a database where the first one of the concurrent search processes to finish will terminate the overall search. We can also model the unreliability of tasks by representing each task by a parallel combination of the task execution and the failure process of the task.

Suppose we are modeling closed (nonrepairable) fault-tolerant systems with permanent faults. Such systems are defined in [15], where they are analyzed by Markov chain techniques. We note that our graphs allow more general distributions of subsystem or component lifetimes than those allowed by the Markov chain techniques. In the SPADE technique, the distribution function attached to a graph node will be the lifetime distribution (or unreliability function) of the component or subsystem it represents. A system consisting of a series combination of components is modeled by parallel graph nodes with type minimum; a parallel combination is modeled as parallel graph nodes with type maximum. Systems with redundant components, some minimum number of which must function, are modeled as k-out-of-n parallel subgraphs.

We can also model the point (instantaneous) availability for the restricted class of repairable systems where each component has an independent repair facility.

D. Implementation of the Graph Analysis

In practice, the calculation of the distribution of a graph is done in two steps. First we decompose the graph, obtaining a tree representation of the graph that shows the series and parallel combinations that formed the graph. The leaves of the tree correspond to nodes of the graph, and each internal tree node represents either a series or parallel combination of its subtrees. Fig. 3 shows a graph and its tree decomposition. When the decomposition is parallel, we label the internal node with the particular interpretation (maximum, minimum, probabilistic, or k-out-of-n) assigned to the set of parallel subgraphs. It is possible to carry out this decomposition in time proportional to the number of nodes in the resulting tree. For a description of such a linear algorithm for decomposing any transitive series-parallel graph into a binary tree, see [26]. The actual algorithm used in the SPADE program is described in [19].

Once we have the tree representation of a graph, the second step is to calculate the graph's distribution according to the definitions in Section II-B. We do that by visiting all of the nodes of the tree in postorder, assigning each one a distribution. The distribution of the root node of the tree will be the CDF of the graph.

If a node is a leaf we assign it the distribution of the graph node to which it corresponds. If a node is internal, it represents the series or parallel combination of two or more subgraphs. Because we are traversing the tree in postorder, the children of the node will already contain the distributions of these subgraphs. We apply one of the formulas (1) through (6a), depending on the type of the

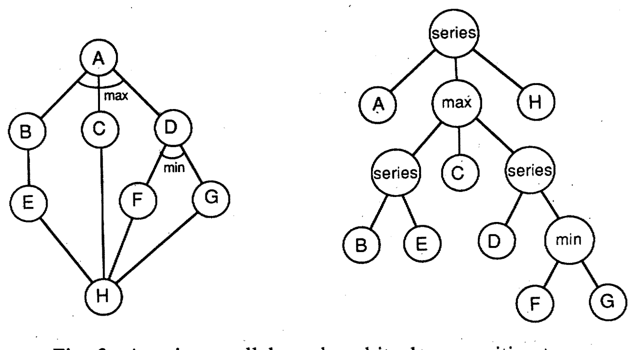


Fig. 3. A series-parallel graph and its decomposition tree.

node, to the distributions assigned to the node's children, and assign the resulting distribution to the node itself. When we reach the root node, we are finished.

If we take as our unit of calculation the multiplication and convolution of distributions, then the time needed to calculate the CDF for a tree node of type series, maximum, minimum, probabilistic, or k-out-of-n with identically distributed subgraphs is $O(n)$. If the type is k-out-of-n with nonidentically distributed subgraphs, the time is at worst (when k is small) $O(n^2)$.

The overall time complexity of the algorithm depends on the representation of distributions, and on the implementation of the various operations done on distributions.

E. Distribution Functions

Up to this point, we have made no assumptions about the character of the CDF's associated with the nodes of our graphs except that they are statistically independent. For CDF's of any form, it would be possible to compute $F(t)$ numerically for the overall graph for any particular value of t. If the type of the CDF's is restricted to be of exponential polynomial form and the parameters are given, we can compute the overall CDF as a function of t.

An exponential polynomial is defined to be an expression of the form

$$\sum_i a_i t^{k_i} e^{b_i t}$$

where the k_i are nonnegative integers and the a_i and b_i are real or complex numbers. Of course, not every exponential polynomial is a valid CDF. For a function $F(t)$ to be the CDF of a nonnegative random variable, it must satisfy the following properties:

$$F \text{ is real-valued.} \tag{7}$$

$$0 \le F(t) \le 1 \; \forall \, t. \tag{8}$$

$$F \text{ is monotone nondecreasing and right continuous.} \tag{9}$$

Note that these properties imply that the real part of each b_i must be less than or equal to zero, and that if $b_i = 0$ then we must have $k_i = 0$. In order for $F(t)$ to be real-valued, complex numbers must occur in conjugate pairs.

That is, whenever $F(t)$ contains the term at^ke^{bt}, if the imaginary part of a or b is nonzero, $F(t)$ must also contain the term $\bar{a}t^ke^{\bar{b}t}$.

The class of exponential polynomials that satisfy these requirements is quite general. In fact, this class is exactly the class of Coxian phase-type distributions [3], [19]. In particular, the CDF of each node can be exponential, hyperexponential, Erlang, or a mixture of Erlang distributions. Because the class of exponential polynomials is closed under the operations of addition, subtraction, multiplication, differentiation, and integration, a series-parallel graph whose nodes have exponential polynomials for CDF's will have an overall CDF that is also an exponential polynomial.

If $F(0) > 0$, then $F(0)$ is a discrete probability mass at the origin. $F(0)$ is the probability that the activity with distribution F takes no time. If an activity represents the failure of a component, it is useful to allow for the possibility that the component is defective to begin with. Also, the distribution of the waiting time in a queueing system usually possesses a mass at the origin [25].

It is possible to have $F(0) = 1$ (F is the "zero" distribution), in which case the activity always takes no time. This is the counterpart of an instantaneous transition in a stochastic Petri net [6]. Nodes with distribution "zero" can be used to specify precedence relations that otherwise would not adhere to our definition of series-parallel (see graphs G1 and G2 in Fig. 2).

The limit of a distribution at infinity represents the probability that an activity ever finishes. If $\lim_{t \to \infty} F(t) < 1$, then F is a "defective" distribution. If an activity represents a numerical algorithm that does not always converge, it is useful to be able to express the probability that the algorithm does not converge. Similarly, if we consider program execution in a failure-prone environment, then we may allow for the possibility of a failure occurring before program completion, so that the program does not complete.

Section IV includes examples that show how distributions with a mass at zero or infinity can be used.

Equations (1)–(6a) for computing the distributions of combinations of subgraphs are valid for distributions with a mass at zero or infinity (or both) as well as for absolutely continuous distributions. In practice, we do not have to implement the formulas in their full generality; instead we rederive them taking into account the exponential polynomial form of the distributions.

III. The SPADE Progam

We have developed a software tool (also called SPADE) that implements the SPADE modeling technique. It is written in C and consists of about 2800 lines of code. SPADE may be used either interactively or in batch mode. The data that must be supplied by the user is the same in either case. When used interactively, SPADE prompts the user for data entry, allowing retry whenever possible if invalid data is entered and ensuring that all required data is entered. In batch mode, the user creates a file that contains data in the form of a simple input language. The SPADE program has proven to be easy to learn and use. Some of its features are:

- It recognizes arithmetic expressions containing any well-formed combination of variables, constants, function names, and the operators +, −, *, /, exponentiation and parentheses. (Variables must be bound to values before graph analysis takes place.)
- It allows users to define arithmetic functions of any number of parameters (including none).
- In addition to recognizing certain built-in distribution functions, it allows user to define distribution functions in any number of parameters (including none). A distribution can be specified by giving a list of exponential polynomial terms or it can be specified to be the distribution function that results from the analysis of some specified graph.
- It contains a convenient shorthand for specifying the common case when the distributions of all subgraphs in a set of parallel subgraphs are identical.
- It allows the user to select what results are printed and, to some extent, the format of the results.
- The input can be split into any number of files. This makes it easy to set up a permanent library of function, distribution, and graph definitions.
- When input is given in the form of block diagrams, fault trees, acyclic Markov chains, or acyclic semi-Markov processes, the model analysis is done using algorithms specialized for these particular models, rather than transforming the models into series-parallel graphs and using the general algorithm. With these specialized algorithms, the analysis is quite efficient.

IV. Examples

This section contains examples of models that can be analyzed using the SPADE technique.

A. Example 1—CPU-I/O Overlap

Fig. 4 shows a series-parallel graph representing one iteration of the program with CPU-I/O overlap considered by Towsley, Chandy, and Browne [24]. In each iteration of the program there are two stages. The first stage is always a CPU burst. The second stage consists of either pure input/output, or input/output that may be overlapped with a second CPU burst. The probability that the second stage consists of CPU-I/O overlap is given by p. The node called *ZERO* is a dummy node having distribution zero. It allows us to have one branch of the *CPU* node lead to a single node, while the other branch leads to a group of nodes to be executed in parallel.

We might like to know how much it helps to allow the overlap. We define the "speedup" to be the ratio of the mean sequential execution time (the time when no overlap is allowed) to the mean parallel execution time. We can use the SPADE program to compute the speedup for various values of p. Fig. 5(a) shows part of an input file for the SPADE program and Fig. 5(b) shows the output produced by SPADE.

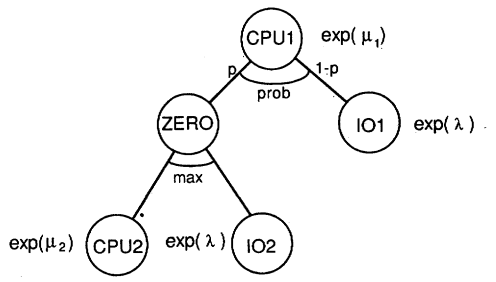


Fig. 4. One iteration of CPU-I/O example.

```
*  by  this point, the variables lambda, mu1
*  and mu2 have been assigned the values
*  1/0.0376,1/0,125 and1/0.14995 and the
*   graph SERIAL has been defined.

graph OVERLAP(p)
cpu1    zero
cpu1    io1
zero    cpu2
zero    io2
end

exit  cpu1  prob
prob  cpu1  zero p
exit  zero  max
dist  cpu1  exp ( mu1)
dist  zero  zero
dist  io1   exp ( lambda)
dist  cpu2  exp ( mu2)
dist  io2   exp ( lambda)
end

expr mean(SERIAL;0.7)
expr mean(OVERLAP(0.7)
func speedup(p)\
      mean(SERIAL;p)/mean(OVERLAP;p)
expr speedup(0.6)
expr speedup(0.7)
expr speedup(0.8)
expr speedup(0.9)
expr speedup(1.0)

bind
mu1   1 / 0.01
end

expr speedup(1.0)
end
```

(a)

```
mean(SERIAL;0.7):  2.7505e-01

----------------------------------------

mean(OVERLAP;0.7):  2.2733e-01

----------------------------------------

speedup(0.6):  1.1845e+00

----------------------------------------

speedup(0.7):  1.2099e+00

----------------------------------------

speedup(0.8):  1.2341e+00

----------------------------------------

speedup(0.9):  1.2570e+00

----------------------------------------

speedup(1.0):  1.2790e+00

----------------------------------------

speedup(1.0):  1.3145e+00
```

(b)

Fig. 5. Input and output for CPU-I/O example. (a) Input. (b) Output.

When p is 0.7, the mean execution time for the serial graph is 0.27505, the mean execution time for the graph with overlap is 0.22733, and the speedup is 1.21. It is interesting to note that even when we have maximum parallelism for this graph (when the branch leading to *IO1* is never taken), the speedup is only 1.28. This is because of the time spent in *CPU1*. When we decreased the mean service time at *CPU1* to 0.01 rather than 0.0376, the speedup with maximum parallelism is increased to 1.31.

As a further experiment, we can add more detail to the model. First, we recognize the fact that the time spent in each CPU may vary, depending on the particular job being done. We can model this by dividing the jobs into n classes and assigning each CPU an n-stage hyperexponential distribution. Second, we assume that the I/O service consists of three sequential phases. The first phase, corresponding to seek time, is assumed to be exponentially distributed with a mass at the origin:

$$F_{\text{seek}}(t) = p_{\text{noseek}} + (1 - p_{\text{noseek}})\,(1 - e^{-\lambda_{\text{seek}} t}).$$

The second phase is the rotational latency phase, and the third phase is the transfer phase; we assume that the time spent in each of these phases in exponentially distributed.

We choose the parameter values so that the mean for the hyperexponential distribution for each CPU node is the same as the mean for the previously used exponential distributions for the nodes, and the mean for the I/O nodes is the same as before. The more detailed model shows a shorter mean execution time for the graph with overlap and a greater speedup for each value of p. This illustrates the fact that the mean of a distribution does not contain all of the information about a distribution.

It took the SPADE program about 0.9 seconds (more than half of which was input/output time) on a lightly loaded Gould CONCEPT™ 32/87 to compute all of the results discussed above.

B. Example 2—Modeling Interprocess Communication

Consider the task graph shown in Fig. 6. This model is based on a model suggested by Kung [11]. Nodes 1, 2, 3, and 4 represent tasks; tasks 1 and 2 are executed on one processor and tasks 3 and 4 are executed on another processor. Results from task 1 must be sent from one processor to the other before task 3 can begin, and similarly for tasks 2 and 4. The time needed for communication between tasks 1 and 3 and tasks 2 and 4 is modeled by the nodes S13 and S24, respectively.

Kung assumed that all of the distributions are exponential and analyzed the graph by converting it into a Markov chain. The SPADE technique allows distributions to be any exponential polynomial. We assigned task 2 the 2-stage Erlang distribution with parameter 0.4, and the rest of the tasks the exponential distribution with parameters 0.3, 0.57, and 0.25 for tasks 1, 3, and 4, respectively. Each communication task is exponentially distributed with mean c. By varying the value of c, we can get a feel for when the cost of communication outweighs the benefits gained from using two processors. The results showed that when c is greater than about 1.25, the communication cost causes the two-processor implementation to take longer (on the average) than if the tasks were all run on a single processor.

We can use the defective distributions allowed by SPADE to model the case where the communication link can fail so that with some probability, the overall program will not finish. If the link used for communication task S13 has failure rate λ_{13} then the completion time distribution for S13 is

$$\left(\frac{1/c}{\lambda_{13} + 1/c}\right)\left(1 - \exp\left[-(\lambda_{13} + 1/c)t\right]\right).$$

The distribution for S14 is computed similarly. When these defective distributions are used, the resulting CDF for the entire graph is defective, and gives both the probability of a link failure before completion of all the tasks as well as the distribution for the time-to-finish in case all

™CONCEPT is a trademark of Gould Inc.

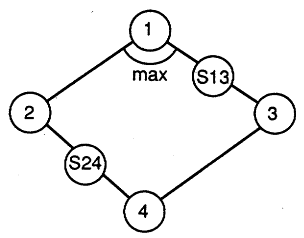


Fig. 6. Communicating tasks.

tasks do complete. If $\lambda_{13} = 0.0001$, $\lambda_{24} = 0.0003$, and $c = 1$, the probability of a link failure before completion of the graph is 0.0003999.

C. *Example 3—Program Execution with a Possibility of Failure*

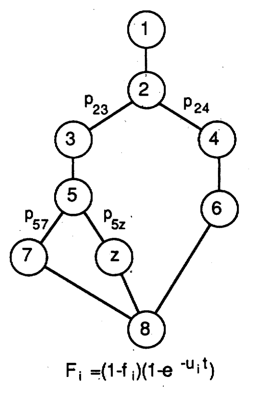
To see how SPADE can be used to analyze the completion time of a program that is subject to software or hardware failure, we consider an example taken from Wei and Campbell [27]. In Fig. 7(a), the nodes in the graph represent segments of a process. Associated with each segment i is the distribution function $F_i(f_i, \mu_i, t) = (1 - f_i)(1 - e^{-\mu_i t})$. The probability of failure during the execution of the segment is $1 - \lim_{t \to \infty} F(f_i, \mu_i, t) = f_i$. All branching in the graph is probabilistic, and the label p_{ij} on the edge leading from node i to node j gives the probability that after the completion of segment i the branch to segment j is taken.

In [27], a formula is given for approximating the overall failure probability. Using the SPADE technique, the result function F for the overall graph gives the CDF for the completion time of the entire process. The mass at infinity of this CDF gives the probability p that a failure occurs before the whole process completes. The distribution F/p is the CDF of the process completion time given that a failure did not occur.

We used SPADE to analyze this graph for the same two sets of values for the probabilities on the edges and failure probabilities as in [27]. We assigned the μ_i arbitrary values, since the original example did not contain execution-time parameters. Fig. 7(b) compares the exact results from SPADE to the approximations obtained by the method used in [27]. As expected, the approximation is better when the individual failure probabilities are smaller.

D. *Example 4—Reliability of an Aircraft Flight Control System*

To illustrate the use of k-out-of-n parallelism and see how the SPADE technique can be used to analyze a fault tree model, we consider example problem 7 in Appendix G of [1]. This problem models an aircraft flight control system. The system contains three inertial reference sensors (IRS) and three pitch rate sensors (PRS), that monitor the status of the aircraft. All of the sensors are connected to each of four computer systems (CS). The computer systems independently collect information from the sensors and process the information. The computers are connected to each other and to three secondary actuators (SA) through four identical bus systems (BS).



(a)

	dataset 1	dataset 2
SHARPE Result	0.2559	0.0283
Wei & Campbell Approximation	0.2864	0.0286

(b)

Fig. 7. A program that may fail. (a) Graph of program. (b) Comparison of results.

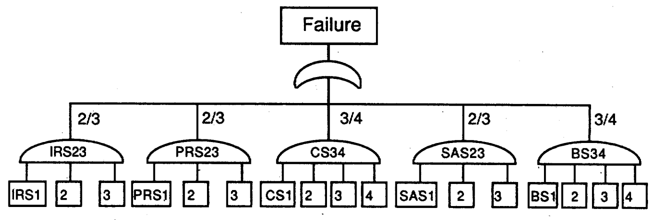


Fig. 8. Aircraft control system.

In order for the entire system to function (so that the aircraft remains airborne) at least two of each type of component must be functioning. A fault tree for this system is given in Fig. 8. It took the SPADE program less than half a second to compute the failure time CDF and the probability of failure for ten values of t.

E. *Example 5—Instantaneous Availability*

If a system is composed of components, each having an independent repair facility, the SPADE technique can be used to compute the instantaneous availability of the system. Consider the series-parallel system of components pictured in Fig. 9. This is the example presented in [14], where an approximation method is given for computing the steady state unavailability of a series-parallel system. Using our model, we can compute the steady state unavailability exactly, and in addition we compute the transient unavailabilities.

Assume that each component is subject to failure, and has its own independent repair facility. If the time-to-failure of component i is exponentially distributed with failure rate λ_i, and the time-to-repair is exponentially distributed with repair rate μ_i, then the instantaneous availability is [25]

$$A_i(t) = \frac{\mu_i}{\lambda_i + \mu_i} + \frac{\lambda_i}{\lambda_i + \mu_i} e^{-(\lambda_i + \mu_i)t}.$$

As t approaches infinity, $A_i(t)$ approaches the steady-state availability. If $\mu_i = 0$ (no repair), $A_i(t)$ reduces to the reliability (as a function of time) of the component.

Let the distribution function associated with component i be $U_i(t) = 1 - A_i(t)$. This distribution represents the instantaneous unavailability of the component and is in

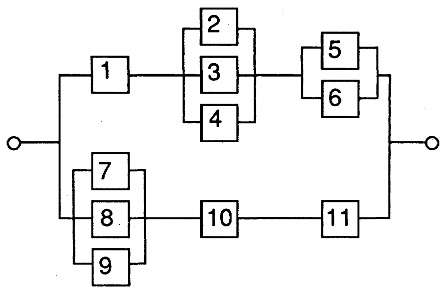


Fig. 9. Availability block diagram.

exponential polynomial form with a mass at infinity. We want to compute the instantaneous unavailability of the system as a whole. For subsystems in parallel, we must take the product of the component unavailabilities (the system is unavailable only when all parallel subsystems are unavailable). This is the "maximum" combination. For a series of components, the availability is the product of the component availabilities (the system is available only when all subsystems are available). Thus, the unavailability of the system is exactly the "minimum" combination of the components.

Because the combining operations are exactly "maximum" and "minimum," we can use SPADE to compute $U(t)$, the instantaneous unavailability for the system as a whole. By taking the limit of $U(t)$ as t approaches infinity, we obtain the steady-state system unavailability, and by setting all $\mu_i = 0$, we obtain system unreliability as a function of the mission time t. Note that μ_i may be zero for some or all of the components, and we still obtain the instantaneous unavailability for the overall system.

We used SPADE to compute the unavailability for this system using the same parameters as in [14]. Those parameters are $\lambda_5 = \lambda_6 = 0.005$, $\lambda_1 = 0.001$, $\lambda_i = 0.01$ for all other i, $\mu_5 = \mu_6 = 1/6$, $\mu_1 = \mu_{10} = \mu_{11} = 1/5$, and $\mu_i = 1/7.5$ for all other i. The steady-state unavailability is computed to be $5.7430 * 10^{-4}$. It should be noted that the approximation given in [14] is in error. The approximation as computed using the method in [14] should be $7.1021 * 10^{-4}$.

Because SPADE distributions can contain complex numbers, we can allow the failure or repair time distributions to be nonexponential. Suppose the failure time distribution for a component i is 2-stage Erlang with parameter $2\lambda_i$ and the repair time distribution is exponential with rate μ_i. Then its instantaneous availablility is given by:

$$A_i(t) = \frac{\mu_i}{\lambda_i + \mu_i} + \frac{\lambda_i}{\lambda_i + \mu_i}\left(\frac{\theta_{i1} - \lambda_i - \mu_i}{\theta_{i1} - \theta_{i2}} \cdot e^{-\theta_{i1}t} + \frac{\theta_{i2} - \lambda_i - \mu_i}{\theta_{i2} - \theta_{i1}} e^{-\theta_{i2}t}\right)$$

where

$$\theta_{i1}, \theta_{i2} = \frac{4\lambda_i + \mu_i \pm \sqrt{\mu_i(\mu_i - 8\lambda_i)}}{2}.$$

If $\mu_i < 8\lambda_i$, the above function will contain complex numbers, a situation easily handled by SPADE.

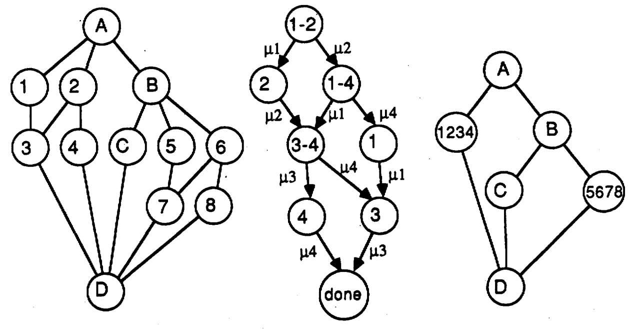


Fig. 10. Analyzing a non-series-parallel graph. (a) Non-series-parallel graph. (b)Lower level: Markov chain. (c) Upper level: series-parallel graph.

F. Example 6—A Hierarchical Model

Suppose we have the task graph shown in Fig. 10(a). Assume that all tasks must run to completion. This graph is non-series-parallel, so we cannot use the series-parallel graph model directly to find the distribution of its completion time. However, if the distributions of the nodes in the non-series-parallel parts are exponential, we can make use of the ability of the SPADE technique to combine models hierarchically. We can extract the non-series-parallel portions of the graph and model them by Markov chains. Then we can replace each non-series-parallel portion of the graph by a single node assigned the CDF of the time to absorption in the corresponding Markov chain. This will give us an exact answer.

Consider the subgraph containing nodes 1, 2, 3, and 4. The Markov chain in Fig. 10(b) is a representation of this system. Each state in the Markov chain is a list of those nodes that are currently running but have not yet completed. The time to reach the absorbing state *done* in the Markov chain is exactly the time-to-completion of the subgraph. Since the subgraph containing nodes 5, 6, 7, and 8 is structurally the same as the subgraph containing nodes 1, 2, 3, and 4, it can be represented by the same Markov chain, with possibly different values for the rates μ_i. Fig. 10(c) shows the graph with the non-series-parallel parts replaced by the single nodes 1234 and 5678.

V. Conclusions and Future Work

We have developed a modeling technique for analyzing stochastic activity networks having series-parallel structure. We allow node execution times to have general exponential polynomial form and allow these distributions to have a mass at the origin and a mass at infinity. We allow several interpretations of parallel subgraphs, including the possibilities of required completion of one, all, or k of the n subgraphs. Parallel subgraphs are allowed to have nonidentical (but independent) distributions.

This technique allows us to model a wide-ranging set of applications including the execution time analysis of concurrent programs, program execution in a failure-prone environment, reliability analysis of nonrepairable fault-tolerant systems, and availability analysis of a class of repairable systems.

Several generalizations of the techniques discussed in this paper have been investigated or are under investigation. The SPADE technique has been extended to include irreducible Markov chains and cyclic Markov chains with absorbing states. With this accomplished, we are able to analyze models of the type discussed in [20]. In order to obtain the symbolic CDF for the time to absorption in a cyclic Markov chain, we need to obtain the eigenvalues of its generator matrix. In general, these eigenvalues may be complex, but this does not pose a difficulty since SPADE distributions allow complex numbers. However, the eigenvalue approach has numerical problems, particularly for large chains. We have a version of the SPADE program that allows cyclic Markov models.

References

[1] S. Bavuso, P. Peterson, and D. Rose, "Care III model overview and user's guide," NASA Tech. Memo. 85810, June 1984.

[2] E. G. Coffman, *Computer and Job/Shop Scheduling*. New York: Wiley, 1976.

[3] D. R. Cox, "The use of Complex probability in the theory of stochastic processes," *Proc. Cambridge Philosophical Soc.*, vol. 51, pp. 313–319, 1955.

[4] H. E. David, *Order Statistics*. New York: Wiley, 1981.

[5] B. Dodin, "Bounding the project completion time distribution in PERT networks," *Oper. Res.*, vol. 33, no. 4, pp. 862–881, July–Aug. 1985.

[6] J. B. Dugan, K. S. Trivedi, R. M. Geist, and V. F. Nicola, "Extended stochastic Petri nets: Analysis and applications," in *PERFORMANCE '84*. Paris: North-Holland, Dec. 1984.

[7] J. L. Fleming, "Relcomp: A computer program for calculating system reliability and MTBF," *IEEE Trans. Rel.*, vol. R-20, pp. 102–107, Aug. 1971.

[8] H. Frank, "Shortest paths in probabilistic graphs," *Oper. Res.*, vol. 17, no. 4, pp. 583–599, July–Aug. 1969.

[9] H. Hartley and A. Wortham, "Statistical theory for PERT critical path analysis," *Management Sci.*, vol. 12, no. 1, pp. B-469–481, June 1966.

[10] W. Kleinoder, "Evaluation of task structures for a hierarchical multiprocessor system," in *Proc. Int. Conf. Modeling Techniques and Tools for Performance Analysis*, May 1984.

[11] K. C.-Y. Kung, "Concurrency in parallel processing systems," Ph.D. dissertation, Dep. Comput. Sci., Univ. California, Los Angeles, 1984.

[12] E. L. Lawler, "Sequencing jobs to minimize total weighted completion time subject to precedence constraints," *Ann. Discrete Math.*, vol. 2, pp. 75–90, 1978.

[13] J. Martin, "Distribution of the time through a directed acyclic network," *Oper. Res.*, vol. 13, pp. 44–66, 1965.

[14] M. Modarres, "A method of predicting availability characteristics of series-parallel systems," *IEEE Trans. Rel.*, vol. R-33, no. 4, pp. 308–312, Oct. 1984.

[15] Y.-W. Ng and A. Avizienis, "A model for transient and permanent fault recovery in closed fault-tolerant systems," in *Proc. 1976 Int. Symp. Fault-Tolerant Computing*, June 1976.

[16] L. Ringer, "Numerical operators for statistical PERT critical path analysis," *Management Sci.*, vol. 16, no. 2, pp. B-136–143, Oct. 1969.

[17] J. Riordan, *Combinatorial Identities*. New York: Wiley, 1968.

[18] J. T. Robinson, "Some analysis techniques for asynchronous multiprocessor algorithms," *IEEE Trans. Software Eng.*, vol. SE-5, pp. 24–31, Jan. 1979.

[19] R. Sahner, "A hybrid, combinatorial-Markov method of solving performance and reliability models," Ph.D. dissertation, Dep. Comput. Sci., Duke Univ., 1986.

[20] R. D. Schlichting, "A technique for estimating performance of fault-tolerant programs," *IEEE Trans. Software Eng.*, vol. SE-11, pp. 555–563, June 1985.

[21] M. Shooman, *Probabilistic Reliability, An Engineering Approach*. New York: McGraw-Hill, 1968.

[22] C. Sigal, A. Pritsker, and J. Solberg, "The stochastic shortest route problem," *Oper. Res.*, vol. 28, no. 5, pp. 1122–1129, Sept.–Oct. 1980.

[23] K. Takamizawa, T. Nishizeki, and N. Saito, "Linear-time computability of combinatorial problems on series-parallel graphs," *JACM*, vol. 29, no. 3, pp. 623–641, July 1982.

[24] D. F. Towsley, J. C. Browne, and K. M. Chandy, "Models for parallel processing within programs," *Commun. ACM*, vol. 21, no. 10, pp. 821–831, Oct. 1978.

[25] K. S. Trivedi, *Probability and Statistics with Reliability, Queuing and Computer Science Applications*. Edgewood Cliffs, NJ: Prentice-Hall, 1982.

[26] J. Valdes, R. E. Tarjan, and E. L. Lawler, "The recognition of series-parallel digraphs," *Siam J. Comput.*, vol. 11, no. 2, pp. 298–313, 1982.

[27] A. T. Wei and R. H. Campbell, "Construction of a fault tolerant real-time software systems," Univ. Illinois, Tech. Rep. UIUCDCS-R-80-1042, Dec. 1980.

Robin A. Sahner received the B.S. degree in mathematics and the B.A. degree in music from the University of Hartford, Hartford, CT, in 1976 and the A.M. degree in mathematics and the Ph.D. degree in computer science from Duke University, Durham, NC, in 1978 and 1986, respectively.

Since 1981, she has been a Computer Scientist at Gould Computer Systems Division in Urbana, IL, where her work is concerned with network and interprocessor communications and secure systems. Her research interests include reliability and performance analysis and software engineering.

Kishor S. Trivedi (M'86) received the B.Tech. degree from the Indian Institute of Technology, Bombay, and the M.S. and Ph.D. degrees in computer science from the University of Illinois, Urbana-Champaign.

He is the author of a widely used text, *Probability and Statistics with Reliability, Queuing and Computer Science Applications* (Englewood Cliffs, NJ: Prentice-Hall). Both the text, and his related research activities, have been focused on establishing a unified mathematical modeling foundation for computing system reliability and performance evaluation. Presently, he is a Professor of Computer Science and Electrical Engineering at Duke University, Durham, NC. He has served as a Principal Investigator on various AFOSR, ARO, IBM, NASA, NIH, and NSF funded projects and as a consultant to industry and research laboratories.

Dr. Trivedi is an Editor of the IEEE TRANSACTIONS ON COMPUTERS.