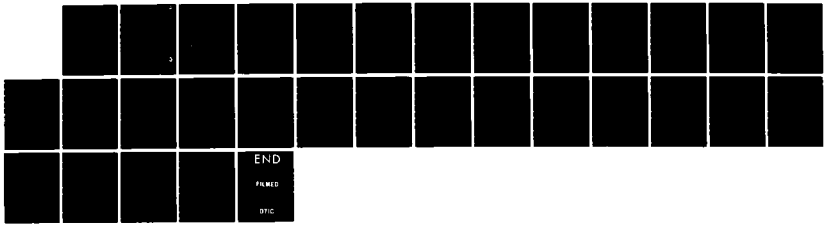
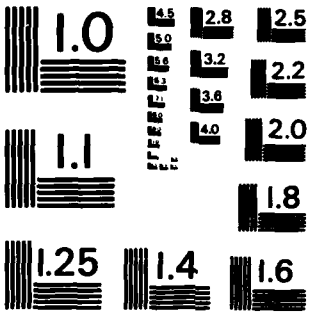
AD-A148 458 SPADE: SERIES-PARALLEL DIRECTED ACYCLIC GRAPH EVALUATOR (U) WISCONSIN UNIV-MADISON MOTOR BEHAVIOR LAB R SAHNER ET AL. NOV 84 CS-1984-15 AFOSR-TR-84-1096 1/1

UNCLASSIFIED AFOSR-84-0132 F/G 9/2 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS - 1963 - A

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION: UNCLASSIFIED	1b. RESTRICTIVE MARKINGS:
2a. SECURITY CLASSIFICATION AUTHORITY:	3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE:	
4. PERFORMING ORGANIZATION REPORT NUMBER(S): CS-1984-15	5. MONITORING ORGANIZATION REPORT NUMBER(S): AFOSR-TR- 84-1096
6a. NAME OF PERFORMING ORGANIZATION: Duke University; 6b. OFFICE SYMBOL (If applicable):	7a. NAME OF MONITORING ORGANIZATION: Air Force Office of Scientific Research
6c. ADDRESS (City, State and ZIP Code): Department of Computer Science, Durham NC 27706	7b. ADDRESS (City, State and ZIP Code): Directorate of Mathematical & Information Sciences, Bolling AFB DC 20332-6448
8a. NAME OF FUNDING/SPONSORING ORGANIZATION: AFOSR; 8b. OFFICE SYMBOL (If applicable): NM	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER: AFOSR-84-0132
8c. ADDRESS (City, State and ZIP Code): Bolling AFB DC 20332-6448	10. SOURCE OF FUNDING NOS. — PROGRAM ELEMENT NO.: 61102F; PROJECT NO.: 2304; TASK NO.: A5; WORK UNIT NO.:

11. TITLE (Include Security Classification): SPADE: SERIES-PARALLEL DIRECTED ACYCLIC GRAPH EVALUATOR

12. PERSONAL AUTHOR(S): Robin Sahner and Kishor S. Trivedi

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Yr., Mo., Day)	15. PAGE COUNT
Technical	FROM _____ TO _____	NOV 84	29

16. SUPPLEMENTARY NOTATION:

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB. GR.	

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

A model for the stochastic analysis of directed acyclic graphs is developed. These graphs represent node-activity networks where the distribution function associated with a node is assumed to be a mixture of Erlangs. The distribution function of the graph execution time is computed in a semi-symbolic form. Applications of the model for the evaluation of concurrent program execution time and to the reliability analysis of fault-tolerant systems are discussed.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS ☐	UNCLASSIFIED

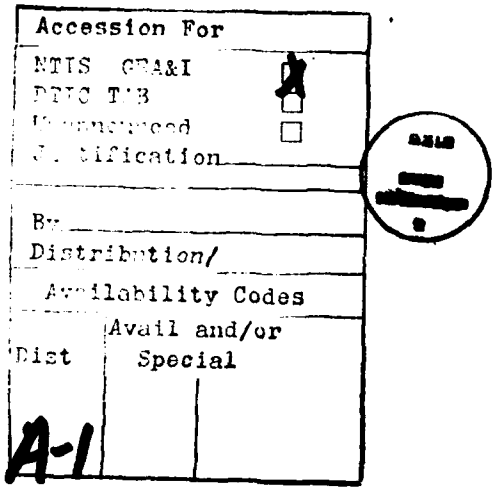
22a. NAME OF RESPONSIBLE INDIVIDUAL	22b TELEPHONE NUMBER (Include Area Code)	22c. OFFICE SYMBOL
MAJ Brian W. Woodruff	(202) 767-5027	NM

DD FORM 1473, 83 APR — EDITION OF 1 JAN 73 IS OBSOLETE. — UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

CS-1984-15

SPADE: Series-Parallel Directed Acyclic Graph Evaluator

Robin Sahner
Kishor S. Trivedi
Department of Computer Science
Duke University
Durham, N. C. 27706



SPADE: Series-Parallel Directed Acyclic Graph Evaluator

Robin Sahner and Kishor S. Trivedi

Department of Computer Science
Duke University
Durham, N. C. 27706

AFOSR-84-0132

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.

Abstract

A model for the stochastic analysis of directed acyclic graphs is developed. These graphs represent node-activity networks where the distribution function associated with a node is assumed to be a mixture of Erlangs. The distribution function of the graph execution time is computed in a semi-symbolic form. Applications of the model for the evaluation of concurrent program execution time and to the reliability analysis of fault-tolerant systems are discussed.

1. *INTRODUCTION*

The demand for computing capacity has sustained a high growth rate. Coupled with technological advances, this demand has implied an increased interest in multiple and distributed processing systems.

Many interesting problems in the design and analysis of such systems need to be solved in order to provide their designers and users with insights and tools for system evaluation. Besides the need to be able to derive measures of system effectiveness for a given system and its associated parameters, there is a need to provide techniques for selecting a set of parameter values in order to optimize system effectiveness in a given setting.

Both simulation and analytic models have been used for system effectiveness evaluation and optimization. Simulation models tend to be more credible but more expensive than analytic models. In order to improve the credibility of analytic models, there is a need to consider more complex models. Most current work on performance analysis of parallel and distributed systems by means of analytic models may be classified as either *program-centered* (transaction-centered) or *resource-centered* analysis.

In resource-centered analysis, system resources are modeled in great detail while a relatively simple model of transaction behavior is assumed. Product-form queueing networks have been used to analyze computer systems [1,20], communication networks, and computer-communication networks [10]. The existence of a product-form solution implies a relatively efficient numerical procedure to obtain the solution. However, real system behavior rarely satisfies the necessary assumptions of a product-form network. For example, transactions with internal concurrency will violate product-form [7,8,19]. Many authors have studied approximate and exact solution techniques for solving queueing models where programs are allowed to overlap their

computation with their own input-output operations [7]. We have recently generalized and studied more general types of concurrency in transactions sharing system resources [8].

In program-centered analysis, a relatively simple model of system resources is assumed while the characteristics of transactions are modeled in great detail. Assuming that transactions possess internal concurrency and randomness, the problem is that of analyzing a stochastic activity network [5,6]. At least three approaches to the analysis of such networks can be identified: path analysis, Markov chain technique, and stochastic Petri net technique.

The path analysis technique first computes the distribution of time to traverse each path. From these path times, the overall execution time can be obtained exactly for certain special types of graphs [16]. But in the general case, overlapping paths exist and hence one can only obtain an approximation (or bounds) for the overall execution time [6]. In any case, for complex graphs the number of paths can be rather large, making the technique computationally expensive.

The second approach is to convert the given graph into a continuous-time Markov chain [11,12]. Besides the restriction imposed by the exponentially distributed node times, this approach quickly leads to an explosion in the state-space of the Markov chain.

The third approach constructs a Petri net equivalent of the given graph. Ramamoorthy and Ho use this approach in the case that node times are deterministic [15]. Molloy considers exponentially distributed node times and converts the Petri net into a Markov chain for analysis [13]. We have allowed the node times to be generally distributed in our Extended Stochastic Petri Net (ESPN) Model [3]. Whenever possible, the ESPN is automatically converted into a Markov chain or a semi-Markov process. If neither of these approaches succeeds then the ESPN is evaluated using Monte-Carlo simulation. The first two approaches for the solution of an ESPN can lead to large state

spaces, while the third approach can be time consuming due to the inherent speed limitations of a simulation model.

- The approach developed in this paper falls into the path analysis category. It avoids the pitfalls of the large state space, enumeration of paths, and restrictive distributional assumptions. This freedom is gained at the expense of considering a restricted class of graphs and a relatively mild restriction on the distribution of node times.

In this paper we consider the analysis of node activity networks which are series-parallel graphs [4]. With each node in the graph is associated a distribution which is a mixture of Erlangs. The distribution function of the total time to traverse the graph is studied. A program called SPADE (*Series-Parallel Directed* acyclic graph *Evaluator*) has been written to compute this distribution. Applications of the use of our model include performance analysis of concurrent programs and reliability analysis of non-repairable fault-tolerant systems.

Robinson [16] and Kleinoder [9] have considered similar graphs for the performance analysis of concurrent programs. Kleinoder's approach differs from ours in that he performs numerical convolutions and other such operations on empirical distributions. Thus his approach avoids any distributional assumptions. However, our approach yields results in semi-symbolic form and is much faster. In addition, our graphs allow for probabilistic branching and minimum node exit types.

After introducing the model in the next section, we give several examples illustrating the use of our approach in section 3.

2. *THE BASIC MODEL*

This section presents a definition of "series-parallel" graphs, describes how such graphs are interpreted as representing tasks and presents an algorithm for computing the probability distribution for the time needed to complete all of the tasks.

2.1 Series-Parallel Graphs

A series of tasks can be represented by an acyclic directed graph where the nodes represent tasks, each of which has a specified cumulative distribution function (CDF), and the arcs represent task precedence. We restrict our attention to a set of graphs called series-parallel graphs. The literature contains a plethora of definitions for the term "series-parallel"; we define the term as follows.

A finite linear graph is 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 x N, called the set of arcs

c) S is the subset of N containing those nodes which are not the second member of any arc in A (these are the entrance nodes).

d) T is the subset of N containing those nodes which are not the first member of any arc in A (these are the exit nodes).

Suppose $G'=(N',A',S',T')$ and $G''=(N'',A'',S'',T'')$ are nonintersecting graphs.

A graph $G=(N,A,S,T)$ is the series connection of G' and G'' iff

a) $N=N' \cup N''$

b) $A=A' \cup A'' \cup (T' \times S'')$

c) $S=S', T=T''$

A graph G is the parallel connection of G' and G'' iff

a) $N=N' \cup N''$

b) $A=A' \cup A''$

c) $S=S' \cup S'', T=T' \cup T''$

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.

If the nodes in a series-parallel graph represent tasks and the arcs represent an ordering of the tasks, then the graph can be interpreted in the following manner. If the graph was formed by the series combination of graphs G' and G'', then all of the tasks in G' must be finished before the first tasks in G'' may begin. If the graph was formed by a parallel combination then there are three possible interpretations.

maximum

The two subgraphs are executed in parallel. Execution of the graph is over when execution of both subgroups is over.

minimum

The two subgraphs are executed in parallel. Execution of the graph is over when execution of either subgroup is over.

probabilistic

Only one of the subgroups G' and G'' is executed. Each subgroup has associated with it a probability of execution.

Figure 1 shows several examples of series-parallel graphs.

One application of these graphs is to model program execution. If we only consider graphs with probabilistic output nodes then such graphs will model flowcharts of loop-free sequential programs. If we only allow maximum output nodes then the graphs will correspond to the task precedence graphs considered in [2]. Finally, if a graph contains only minimum output nodes then the graph will model the parallel execution of a non-deterministic algorithm [18] 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.

The graphs can also be used to model the lifetime of closed (non-repairable) fault-tolerant systems with permanent faults. Such systems are defined in [14], where they are analyzed by Markov chain techniques. A system consisting of a series combination of components is modeled by parallel graph nodes with type minimum; a parallel combination is modeled by parallel graph nodes with type maximum. We should note that our graphs do allow more general distributions of subsystem or component lifetimes than those allowed by the Markov chain techniques used in [14].

For modeling the lifetime of a k out of n type system, we will appeal to the stage-type lifetime derived in [20; figure 3.30]. Similarly, for hybrid NMR system with imperfect coverage we will use the Coxian phase type expansion derived in [20; figure 5.12].

2.2 Evaluation of Graphs

Any series-parallel graph can be decomposed into a binary tree, where the internal nodes are of type "series" or "parallel", according to the rule which was used to form the graph, and the leaves are the nodes of the graph. The parallel nodes are divided into three types: "maximum", "minimum", and "probabilistic". For the probabilistic nodes, each subtree has associated with it the probability that it will be executed. Figure 2 shows a series-parallel graph and the binary tree associated with it.

The decomposition is not necessarily unique, but all possible decompositions of a graph are equivalent in the sense that the probability distributions as computed by all of the possible binary trees are the same. This is true because "maximum" and "minimum" are associative, and for the probabilistic nodes, multiplication is distributive over addition.

Given the binary tree representation of a series-parallel graph and a CDF for the finish time of each node, one can theoretically calculate the CDF of the finish time for the entire graph. Suppose we have a node A in the tree. If A is a leaf, then A represents a node in the graph, and the CDF of A is the CDF of the node. Now suppose A

is not a leaf, and that its subtrees are B and C. Let F_b be the CDF for the subtree B and F_c be the CDF for the subtree C. The calculation for F_a, the CDF of the subtree rooted at A, depends on what kind of node A is.

1. If A is a "series" node, then it represents the execution of the subtree B followed by the subtree C. F_a is given by:

$$F_a(t) = \int_0^t \int_x^t F_b(x)F_c(s-x)\,ds\,dx$$

2. If A is a "maximum" node, then it represents the maximum of the execution times of subtrees B and C; hence:

$$F_a(t) = F_b(t)F_c(t)$$

3. If A is a "minimum" node, then it represents the minimum of the execution times of subtrees B and C; therefore:

$$F_a(t) = F_b(t) + F_c(t) - F_b(t)F_c(t)$$

4. If A is a "probabilistic" node, only one subtree of A will be executed. Suppose the probability that subtree B is executed is p_b and the probability that subtree C is executed is p_c. Then F_a is given by:

$$F_a(t) = p_b F_b(t) + p_c F_c(t)$$

It would be possible, given any series-parallel graph, to compute numerically the value of the CDF of the entire associated tree given any value of t. If the type of the CDF's is restricted to be of exponential polynomial form and the parameters are given, it is relatively easy to compute the overall CDF.

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

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

Exponential polynomials can be easily shown to be closed under the operations of addition, subtraction, multiplication, differentiation and integration. Because exponential polynomials are closed under these operations, a series-parallel graph whose nodes have exponential polynomials for CDF's will have an overall CDF and pdf which are also

exponential polynomials. In particular, the CDF of each node can be exponential, hyperexponential, Erlang, or a mixture of Erlang distributions.

The process of finding the overall CDF of a series-parallel graph whose nodes have CDF's which are exponential polynomials is easily automated. The program SPADE accepts a specification of such a graph and produces the overall CDF, pdf, mean and variance, and computes the overall CDF for a given range of values.

The input to SPADE consists of any number of of ordered pairs of nodes, which must form a series-parallel graph, an exit type for each node which has more than one immediate successor, and a CDF for each node.

The user has three choices for exit type: maximum, minimum or probabilistic. If the exit type is probabilistic, the user must specify the probabilities. For the user's convenience, SPADE allows the user to specify a distribution type for each node; the types allowed are exponential, general, and "zero".

If the distribution type is exponential, the user need only provide the parameter λ. If the distribution type is general, the user must supply the three parameters a_i, k_i and b_i for each term in the exponential polynomial. Zero nodes represent tasks which take no time. They are included as a matter of convenience. The examples in section 3 include instances where zero nodes are used.

SPADE allows a specified graph to have any number of entrance and exit nodes. If the graph has more than one entrance node, SPADE will implicitly supply a single "zero" entrance node which branches off to each of the specified entrance nodes. SPADE will prompt for the exit type of the added zero node. If the graph has more than one exit node, SPADE will supply a single "zero" exit node which acts as a collector node for all of the specified exit nodes. In the examples below, SPADE-supplied nodes are indicated with dashed lines.

Once all information about the graph has been obtained, the program forms a binary tree corresponding to the series-parallel graph. The graph decomposition

results in a preorder representation of the tree. The program then backs up the preorder representation (traversing the tree in "reverse pre-order"), computing the CDF and pdf of each subtree. Once the CDF is known, the program prompts the user for ranges and increments of time for which the CDF is to be computed.

3. *APPLICATIONS*

This section presents examples of applications of the task graphs defined in section 2. The applications are of two kinds: analysis of concurrent task execution and reliability analysis.

3.1. Concurrent Program Execution Time

Example 1. Suppose a program has two phases that are executed sequentially. The graph of the program is shown in Figure 3. The execution time distribution of the first phase execution time is two-stage Erlang with parameter $\lambda=1$ (the CDF is $1-e^{-t}-te^{-t}$). The second phase time distribution is exponential, also with parameter $\lambda=1$.

Our first input to SPADE is the single ordered pair (A,B). Since there are no parallel nodes in the graph, SPADE does not need to be told any exit types. SPADE prompts for the CDF's of the two nodes. In order to specify the distribution for node A, we tell SPADE that the distribution is of type "general" and input a line corresponding to each term in the CDF:

i	a_i	k_i	b_i
1	1.0	0	0.0
2	-1.0	0	-1.0
3	-1.0	1	-1.0

We tell SPADE that the distribution for node B is of type "exponential", and need only input the value 1 (the parameter of the distribution).

SPADE computes the CDF, pdf, mean and variance of time needed for both phases to complete:

$CDF(t) = 1 - e^{-t} - te^{-t} - 1/2t^2e^{-t}$

$pdf(t) = 1/2t^2e^{-t}$

mean = 3

variance = 3

If we ask SPADE to compute CDF (t) for x between 2 and 10, in increments of 2, the results are

t	*CDF(t)*
2.0	0.3233
4.0	0.7619
6.0	0.9380
8.0	0.9862
10.0	0.9972

Example 2. Next we consider another sequential program which consists of two phases. In this case, the outcome of the first phase determines which of two alternative tasks is executed as the second phase. Thus, the node representing the first phase has a probabilistic exit. This program is illustrated in Figure 4.

The execution time of task 1 (the first phase) is exponentially distributed with mean 1. The execution time of task 2 is 2-stage Erlang distributed with parameter 1 and the execution time of task 3 is exponentially distributed with mean 1. The probability that task 2 will be executed is input as 0.9 while the corresponding probability for task 3 is 0.1. The overall program execution time distribution, as computed by SPADE, is given by:

$CDF(t) = 1 - e^{-t} - te^{-t} - .45t^2e^{-t}$

$pdf(t) = .1te^{-t} + .45t^2e^{-t}$

Mean:2.9

Variance:2.99

Values for CDF(t) are

t	*CDF(t)*
2.0	0.3504
4.0	0.7765
6.0	0.9425
8.0	0.9873
10.0	0.9975

Example 3. We evaluate one iteration of the program with CPU-I/O overlap considered by Towsley, Chandy and Browne [19] and shown in Figure 5. Note the use of a "zero" node, which allows us to have one branch of the CPU1 node be a single node, while the other branch leads to a group of nodes to be executed in parallel. SPADE allows the specification of graphs which have multiple exit nodes.

Assuming μ_1=0.125, μ_2=0.0376, λ=0.0217, and p=0.6, the results from SPADE are:

$$CDF(t) = 1.0 - 1.21e^{-0.0217t} - 0.8581e^{-0.0376t} + 1.1414e^{-0.0693t} - 0.0733e^{-0.125t}$$

The mean and variance of the program execution time are 59.938 and 2124.3147, respectively. Note that, in SPADE, we can specify more general distributions for the nodes.

Example 4. In this example we consider the process communication graph from Kung's thesis [12], and shown in Figure 6. Tasks 1 and 2 are executed on one processor and tasks 3 and 4 on another processor. Communication time between tasks 1 and 3 and tasks 2 and 4 is modeled by the nodes S_{13} and S_{24}. Assuming that μ=0.125 and α=10, the distribution function and the mean and variance of the overall execution time computed by SPADE are:

$$CDF(t) = 1.0 - 2.2349e^{-0.125t} + 0.0294te^{-0.125t}$$
$$- 0.0174t^2e^{-0.125t} + 1.2346e^{-0.25t}$$
$$+ 0.0027e^{-1.25t} - 0.0025e^{-1.375t}$$

The mean and variance are: 28.8364 and 208.1388, respectively.

3.2. Reliability Analysis

We now consider a sequence of examples of the use of SPADE for reliability analysis.

Example 5. Consider a series system with two independent components. If either component fails, the entire system has failed. The "task graph" for modeling the lifetime of this system is shown in figure 7.

Node 1: distribution type - ZERO; exit type: minimum

Node 2: distribution type - EXP; parameter $\lambda_1 = 0.0002$

Node 3: distribution type - EXP; parameter $\lambda_2 = 0.0001$

The CDF and reliability functions from SPADE are

$$CDF(t) = 1 - e^{-0.0003t}$$

$$R(t) = .0003e^{-0.0003t}$$

The MTTF is 3333.3333, with variance 11111111.1.

Example 6. Consider a parallel redundant system with unequal failure rates. The system fails only if both components fail. The "task graph" that models the lifetime of this system is shown in figure 8.

Node 1 has distribution type zero, and exit type maximum. Nodes 2 and 3 are as in Example 5 above.

The solution from SPADE is

$$CDF(t) = 1 - e^{-0.0001t} - e^{-0.0002t} + e^{-0.0003t}$$

$$R(t) = .0001e^{-0.0001t} + .0002e^{-0.0002t} - .0003e^{-0.0003t}$$

The MTTF is 11666.6667.

Example 7. We consider a triple modular redundant (TMR) system. A task graph for the lifetime of this system can be derived based on example 3.24 of [20] and is shown in figure 9a.

Node Z1: distribution type - ZERO; exit type - minimum

Nodes 2,3 and 4: distribution type - EXP; parameter $\lambda = 0.0001$

Node Z5: distribution type - ZERO; exit type - minimum

Nodes 6,7: distribution type - EXP; parameter $\lambda = 0.0001$

The solution from SPADE is

$$CDF(t) = 1 - 3e^{-0.0002t} + 2e^{-0.0003t}$$

$$MTTF = 8333.3333$$

It is possible to simplify the task graph for this example as shown in figure 9b (based on example 3.26 of [20]).

Example 8. We now consider a TMR system with one spare unit. We allow the spare failure rate μ to be different from the failure rate λ of the active unit. We also allow for imperfect coverage. The coverage for an active unit is c_a , while the coverage for the spare is c_s .

An uncovered failure in an active unit immediately leads to a system failure while an uncovered failure in the spare will lead to a system failure after a subsequent active unit failure. If the failure of either an active or spare unit is covered, the system continues to operate as long as any two of the remaining three units are operating.

A "task graph" for this system is shown in figure 10.

Node Z1: distribution type - ZERO;
exit type - minimum

Nodes M7, M11, M15, M19, M22: distribution type - ZERO;
exit type - minimum

Node 5: distribution type - EXP;

parameter μ = 0.0001

exit type - probabilistic,

coverage (arc from 5 to M15) = .98

Node P1: distribution type - ZERO;

exit type- probabilistic

coverage (arc from P1 to M7) = .99

Node E14: distribution type -ZERO;

remaining nodes: distribution type - EXP;

parameter μ = 0.0002

SPADE computes the CDF to be:

$$CDF(t) = 1 - .0233e^{-.0003t} + .0098e^{-.0006t} + 4.7282e^{-.0007t}$$
$$- .0039e^{-.0008t} - 6.3043e^{-.0009t} - 2.9108e^{-.0010t}$$
$$+ 5.0032e^{-.0012t} - 1.5365e^{-.0014t} - .0015te^{-.0007t}$$
$$+ .0012te^{-.0010t} - .0005te^{-.0012t}$$

The MTTF is 3121.1882. Some values for the CDF are as follows.

t	$CDF(t)$
2000.0	0.3853
4000.0	0.7037
6000.0	0.8813
8000.0	0.9577
10000.0	0.9859

4. CONCLUSIONS

We have developed a model for the semi-symbolic computation of the execution time distribution a precedence graph. The distribution function of individual node time is assumed to be a mixture of Erlangs. Several applications of the use of our model are

given.

REFERENCES

[1] Chandy, K.M., Howard, J.H., and Towsley, D.F., "Product Form and Local Balance in Queueing Networks," *JACM*, Vol. 24, pp. 250-263.

[2] Coffman, E.G., *Computer and Job/Shop Scheduling*, John Wiley & Sons, NY., 1976.

[3] Dugan, J.B., Trivedi, K.S., Geist, R.M. and Nicola, V.F., "Extended Stochastic Petri Nets: Analysis and Applications", accepted, PERFORMANCE '84, Paris, December 1984.

[4] Elgot, C.C. and Wright, J.B., "Series-Parallel Graphs and Lattices", *Duke Mathematics Journal*, vol. 26 (1959), pp. 325-338.

[5] Fix, W. and Neumann, K., "Project Scheduling by Special GERT Networks," *Computing* 23 (1979), pp. 299-308.

[6] Gaul, W., "On Stochastic Analysis of Project Networks," in M.A.H. Dempter et al. (eds.), *Deterministic and Stochastic Scheduling*, D. Reidel Publishing Co., 1982.

[7] Heidelberger, P. and Trivedi, K.S., "Queueing Network Models for Parallel Processing with Asynchronous Tasks," *IEEE Trans. on Computers*, November 1982.

[8] Heidelberger, P. and Trivedi, K.S., "Analytic Queueing Models for Programs with Internal Concurrency," *IEEE Trans. on Computers*, January 1983.

[9] Kleinoder, W., "Evaluation of Task Structures for a Hierarchical Multiprocessor System", *Proc. Int. Conf. on Modeling Techniques and Tools for Performance Analysis*, Paris, France, May 1984.

[10] Kleinrock, L., *Queueing Systems, Vol. II: Computer Applications*, John Wiley & Sons, 1976.

[11] Kulkarni, V. and Adlakha, W., "Markov and Markov-Regenerative PERT Networks", Tech. Report, Operations Research and Systems Analysis, Univ. of North Carolina at Chapel Hill, 1984.

[12] Kung, K.C.-Y., "Concurrency in Parallel Processing Systems", Ph.D. Dissertation, UCLA Computer Science Department, 1984.

[13] Molloy, M., "On the Integration of Delay and Throughput Measures in Distributed Processing Models", Ph.D. dissertation, Computer Science Department, UCLA, 1981 .

[14] Ng, Y.-W. and Avizienis, A., "A Model for Transient and Permanent Fault Recovery in Closed Fault-Tolerant Systems," *Proc. 1976 Int. Symp. on Fault-Tolerant Computing*, June 1976.

[15] Ramamoorthy, C.V., and Ho, G.S., "Performance Analysis of Asynchronous Concurrent Systems Using Petri Nets", *IEEE Transactions on Software Engineering*, Vol. SE-6, No. 5, pp. 440-449, September 1980.

[16] Robinson, J.T., "Some Analysis Techniques for Asynchronous Multiprocessor Algorithms," *IEEE Transactions on Software Engineering*, Vol. SE-5, No. 1, January 1979.

[17] Sauer, C.H. and Chandy, K.M., "The Impact of Distributions and Disciplines on Multiprocessor Systems," *CACM*, Vol. 22, pp. 25-34.

[18] Sedgewick, R., *Algorithms*, Addison-Wesley, Reading, Mass., 1983.

[19] Towsley, D.F., Browne, J.C. and Chandy, K.M., "Models for Parallel Processing

within Programs," *CACM*, October 1978.

[20] Trivedi, K.S., ***Probability and Statistics with Reliability, Queueing and Computer Science Applications***, Prentice-Hall, Englewood Cliffs, N.J., 1982.

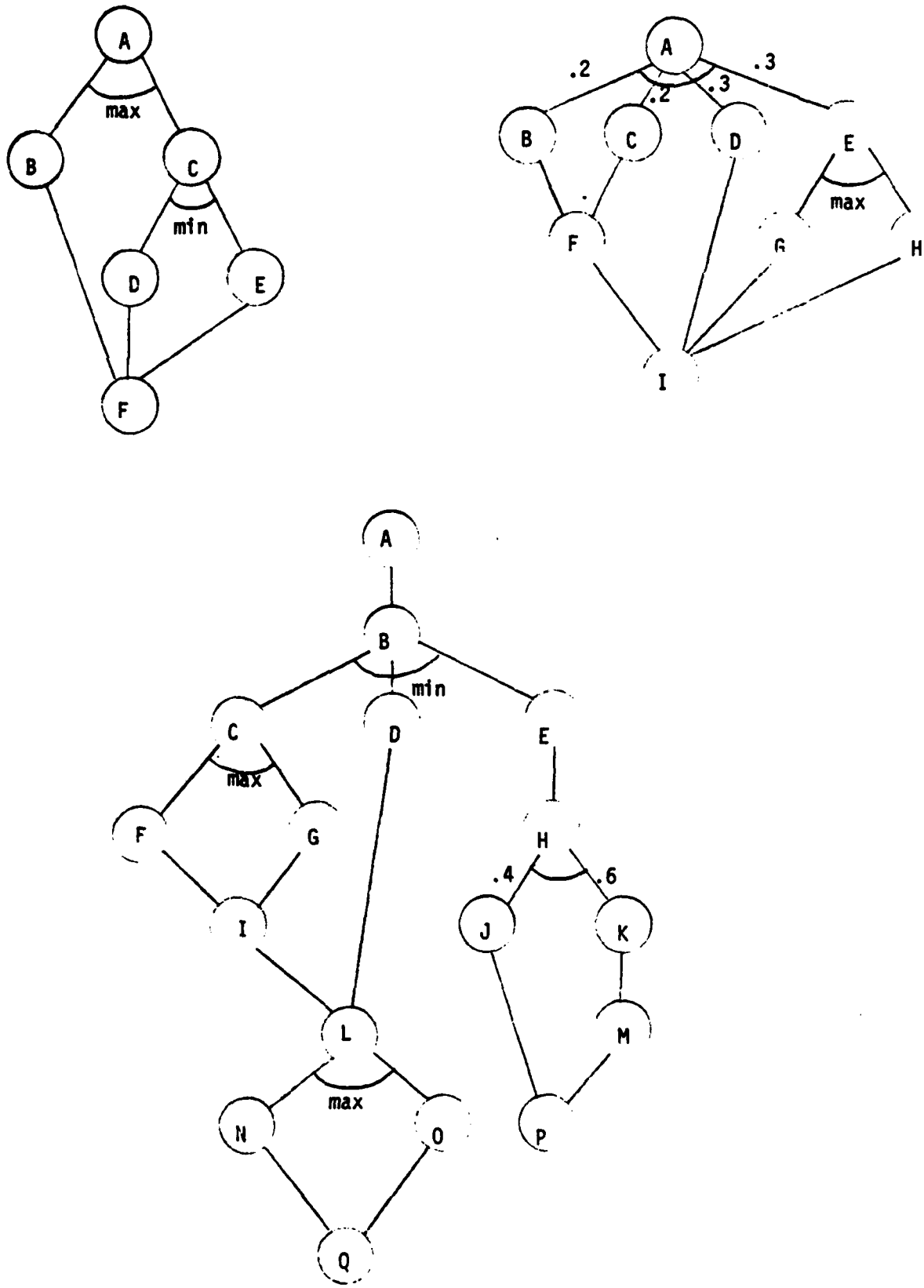


Figure 1 - Examples of Series-Parallel Graphs

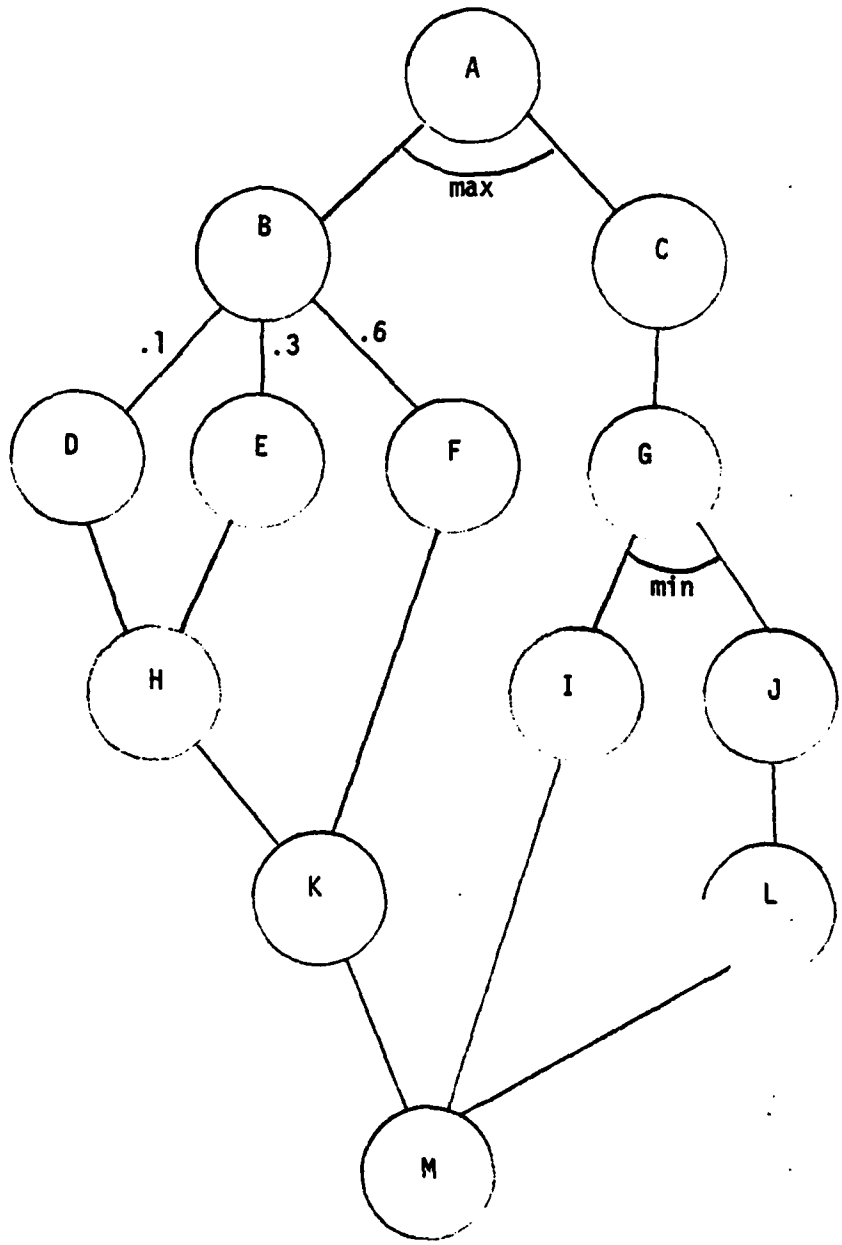


Figure 2a - A Series-Parallel Graph

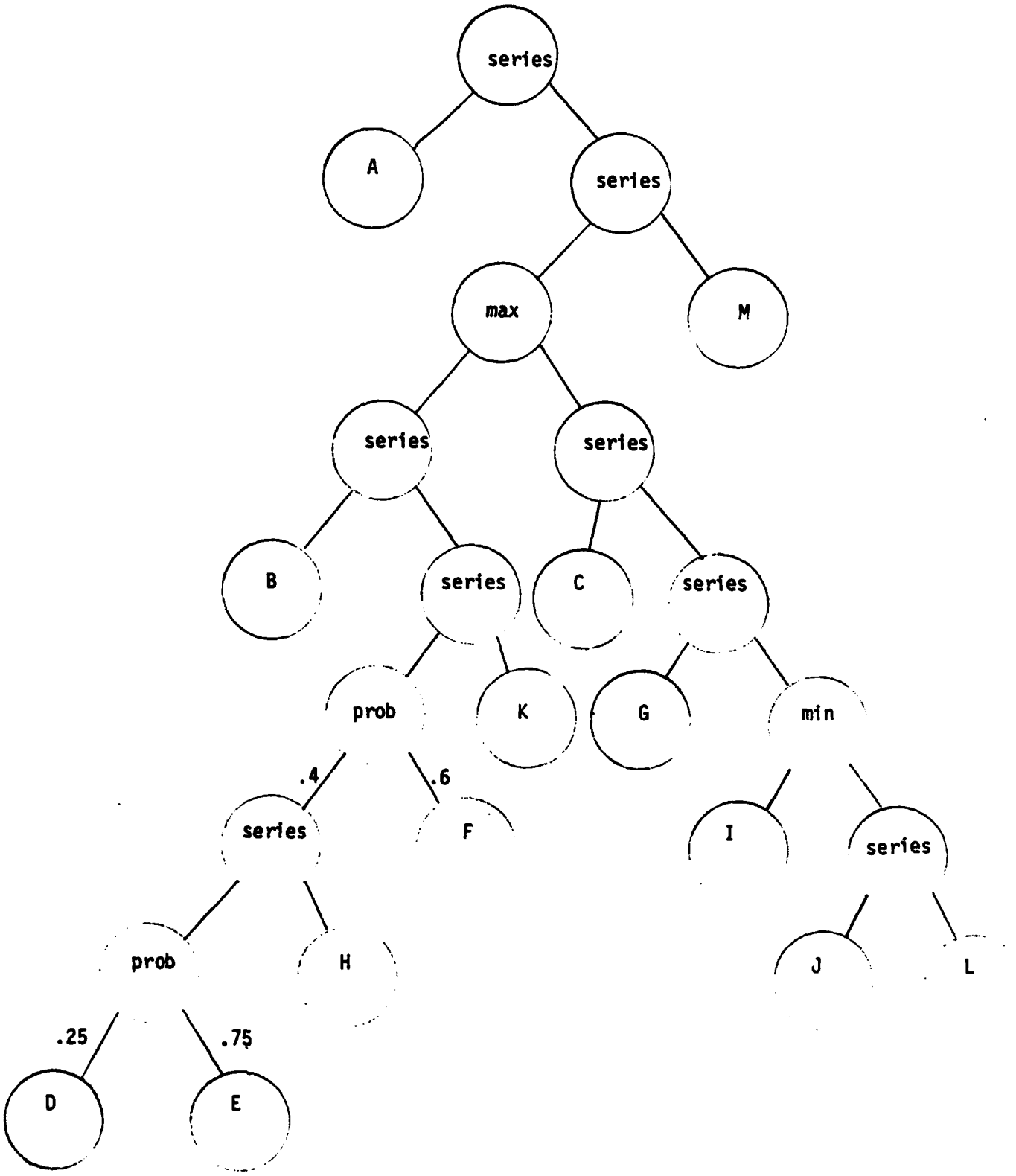


Figure 2b - Binary Tree Decomposition of a S-P Graph

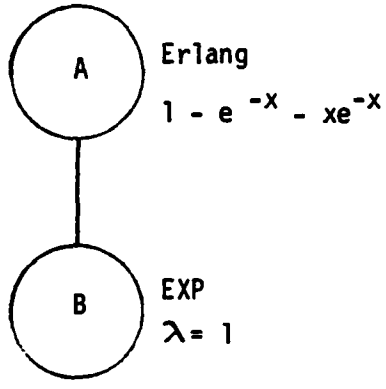


Figure 3 - Tasks Executed in Series

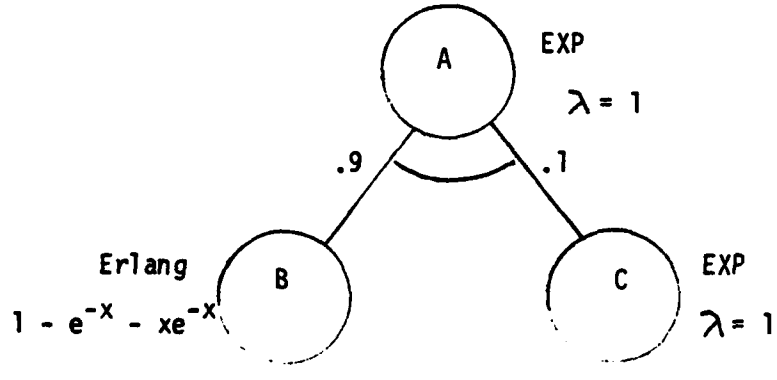


Figure 4 - Series Tasks with a Probabilistic Branch

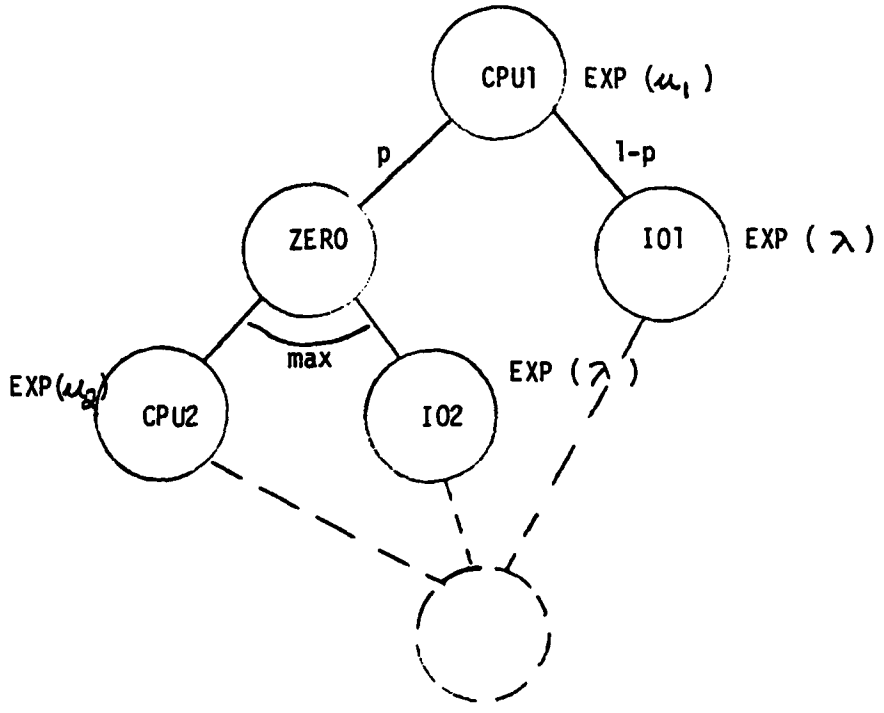


Figure 5 - CPU / IO Overlap

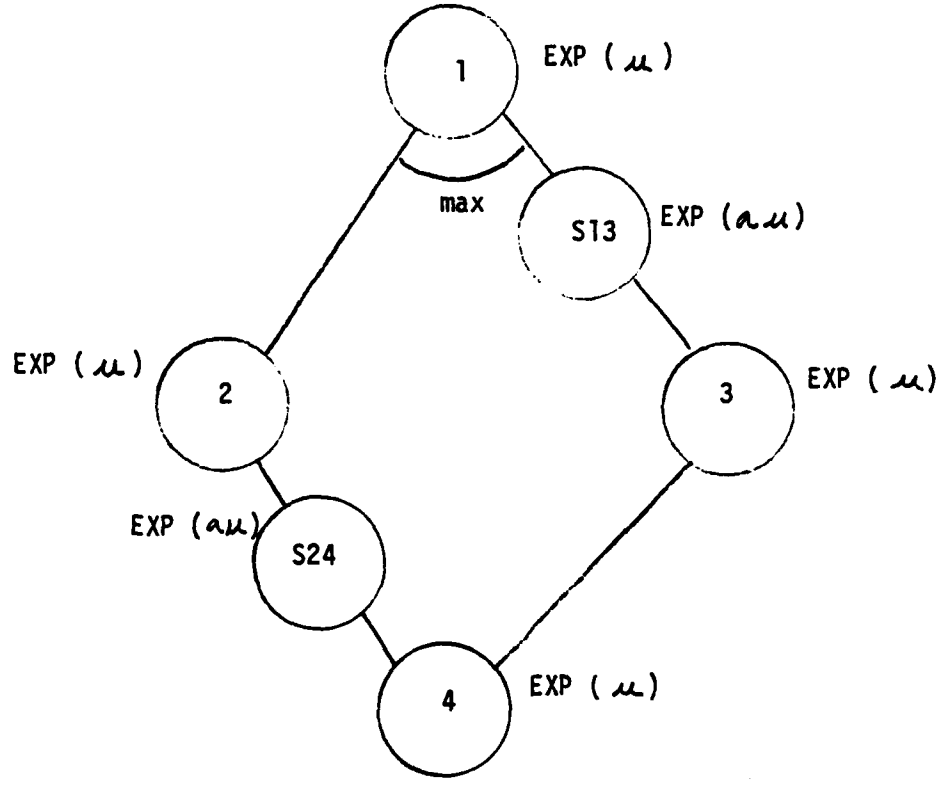


Figure 6 - Example from Kung's Thesis

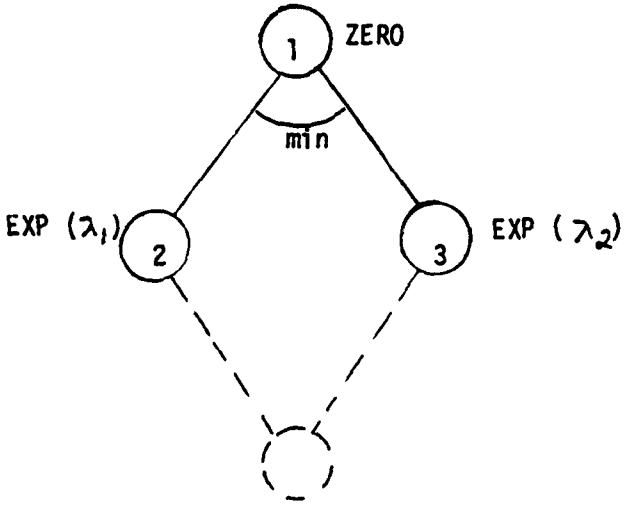


Figure 7 - Components in Series

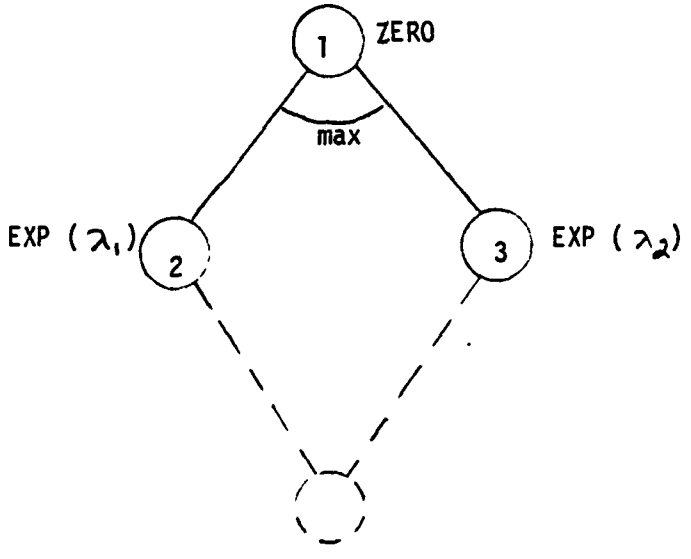


Figure 8 - Components in Parallel

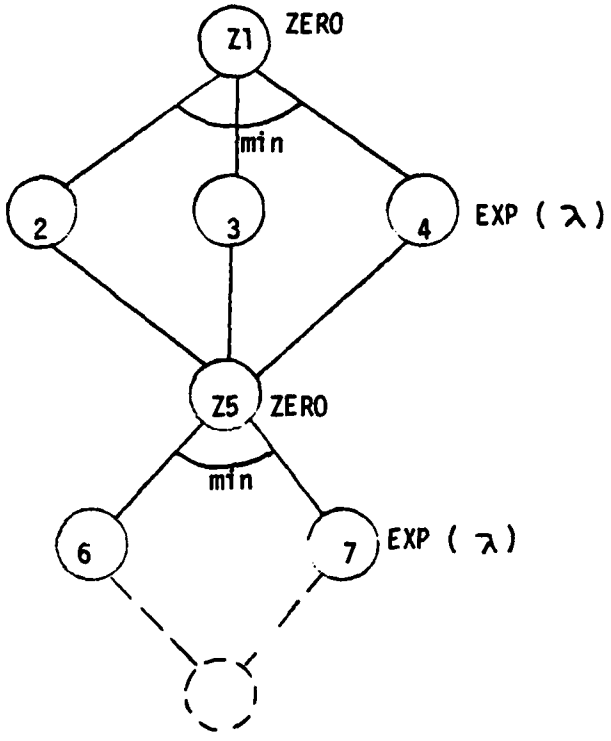


Figure 9a - TMR System

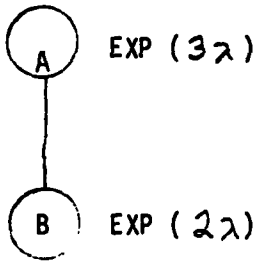


Figure 9b - Simplified TMR Graph

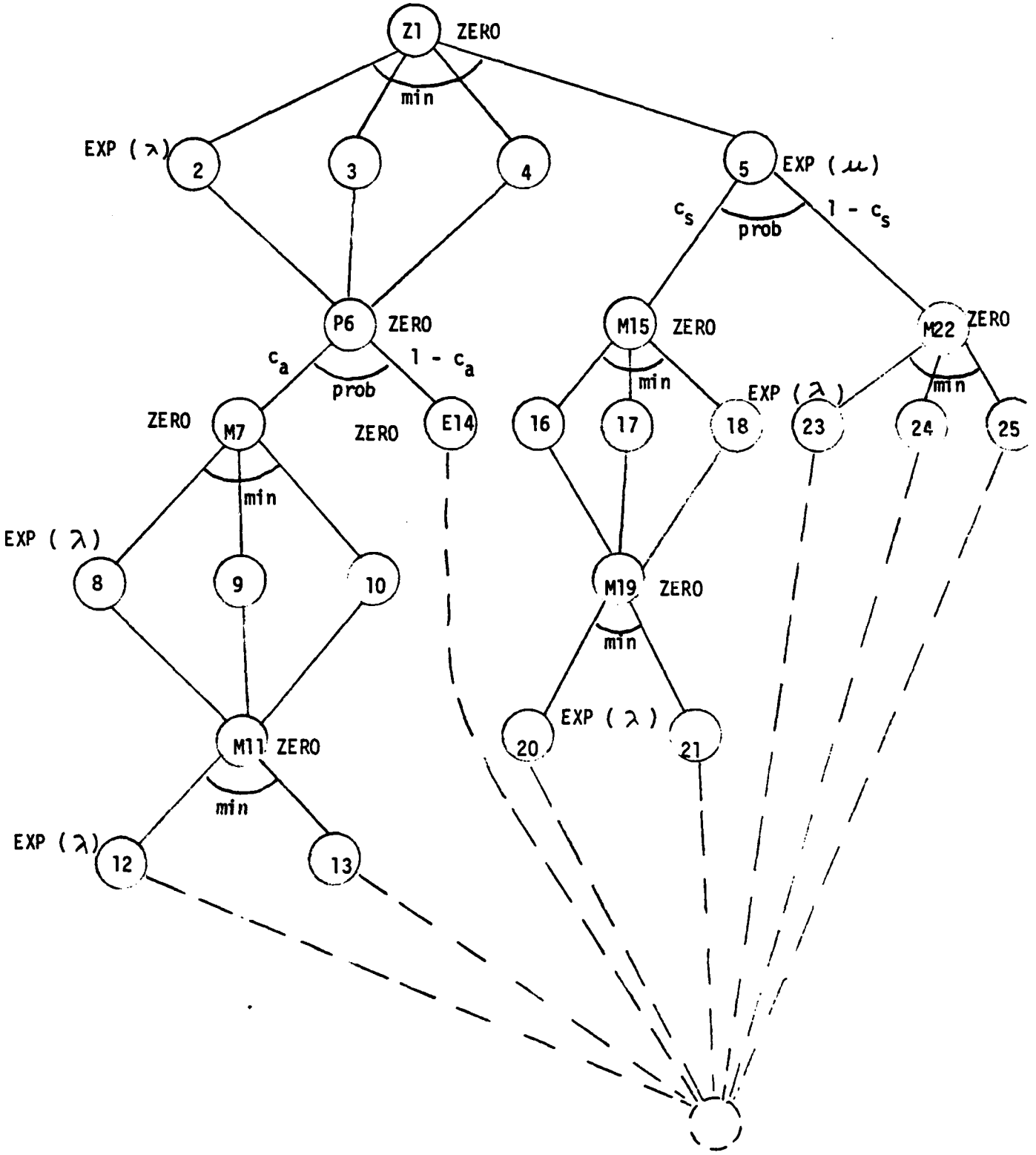


Figure 10a - TMR System with Spare

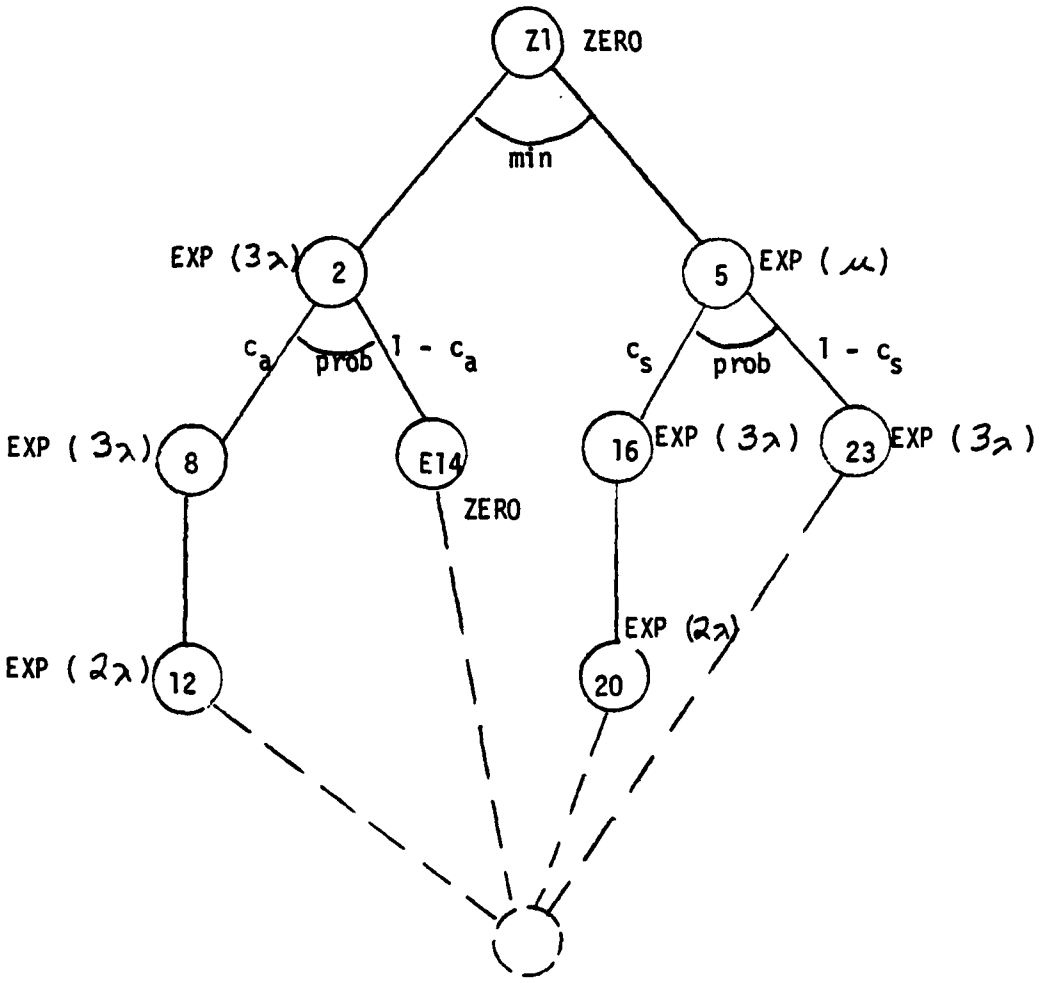


Figure 10b - Simplified Graph for TMR System with Spare

END

FILMED

1-85

DTIC