The Reliability Analysis of Distributed Computing Systems with Imperfect Nodes

MIN-SHENG LIN[1], DENG-JYI CHEN[2] AND MAW-SHENG HORNG[3]

[1]*Department of Information Management, Tamsui Oxford University College, Tamsui, Taipei, Taiwan 25103, Republic of China*
[2]*Institute of Computer Science and Information Engineering, National Chiao-Tung University, Hsin Chu, Taiwan 30050, Republic of China*
[3]*Department of Mathematics Education, National Taipei Teacher College, Taipei, Taiwan 10659, Republic of China*
Email: djchen@csie.nctu.edu.tw

The reliability of a distributed computing system depends on the reliability of its communication links and nodes and on the distribution of its resources, such as programs and data files. Many algorithms have been proposed for computing the reliability of distributed computing systems, but they have been applied mostly to distributed computing systems with perfect nodes. However, in real problems, nodes as well as links may fail. This paper proposes two new algorithms for computing the reliability of a distributed computing system with imperfect nodes. Algorithm I is based on a symbolic approach that includes two passes of computation. Algorithm II employs a general factoring technique on both nodes and edges. Comparisons with existing methods show the usefulness of the proposed algorithms for computing the reliability of large distributed computing systems.

Received July 12, 1994; revised December 3, 1998

1. INTRODUCTION

A typical *distributed computing system* (DCS) consists of processing elements (PEs), memory units, data files and programs. These resources are interconnected through a communication network that dictates how information flows between PEs. Programs residing on some PEs can run using data files stored in other PEs. For successful execution of a program, it is essential that communication links between the PE containing the program and other PEs that have the required data files be operational. *Distributed program reliability* (DPR) is defined as the probability that a distributed program that runs on multiple PEs and needs to communicate with other PEs for remote files will be executed successfully. For example, in the DCS in Figure 1, there are six PEs (n_1, n_2, n_3, n_4, n_5, n_6) and eight communication links (e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8). Program P1 requires data files f1, f2 and f3 to complete its execution, and it is running at node n_2, which holds the file f1. Hence, program P1 must access the remote files f2 and f3. Since file f2 is resident at node n_3 and file f3 is resident at nodes n_4 and n_5, the DPR of program P1 can be formulated as

DPR (for program P1)
= Prob((nodes n_2, n_3, and n_4 are connected)
OR (nodes n_2, n_3, and n_5 are connected)).

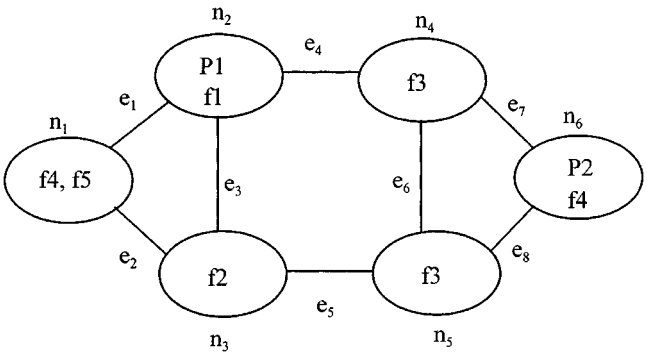

FIGURE 1. A simple DCS.

Many algorithms [1, 2, 3, 4, 5, 6, 7] have been proposed to analyse the reliability of DCSs, but most of these algorithms assume that all nodes in the DCS are perfect. However, in real problems, the nodes as well as edges may fail. In Prasnna Kumar *et al.* [3], the *minimum file spanning tree* (MFST) was proposed to represent the multiterminal connections required to execute a distributed program and a two-pass method for the reliability analysis of a DCS was developed. In this method, all MFSTs are obtained using a breadth-first search method. Since the MFSTs are not

disjoint from each other, once it has found all the MFSTs, the algorithm still requires other terminal reliability evaluation algorithms, such as SYREL [8], to generate the reliability expression. Although Prasnna Kumar's method is elegant, it generates many redundant subgraphs during processing and requires extra time to check and remove them. Thus, it is an inefficient reliability analysis algorithm.

In [5], Ke and Wang proposed an algorithm, ENR/KW (evaluating network reliability/Ke and Wang) which employs a different concept that requires one step to give the reliability expression. The basic idea of the ENR/KW algorithm is to partition the graph directly into a set of disjoint subgraphs. Each disjoint subgraph is generated by maintaining a specific directed graph structure to consider the effect of imperfect nodes. Therefore, the DPR computation can then be carried out by summing all these disjoint probability expressions. This technique represents a one-step approach as there is no need to compute multiterminal connections. Some well known reliability-preserving graph reductions, however, are limited to the specific directed graph structure in the EMR/KW algorithm, and this restriction can increase the complexity of the EMR/KW algorithm that would otherwise use them. The other shortcoming of the EMR/KW algorithm is that it cannot be used to evaluate the reliability of a distributed program running from more than one node.

This paper proposed two algorithms, namely the SM (symbolic method) and the FM (factoring method), for computing the reliability of DCSs with imperfect nodes. SM is a two-pass method like Prasnna Kumar's [3]. SM employs the graph expanding procedure used in the FREA (fast reliability evaluation algorithm [1] to find all MFSTs. Since it has been proved that FREA guarantees that no replicated subgraphs will be generated during the expansion of the computation tree, first pass in SM can also guarantee that no replicated subgraphs will be generated. Therefore, SM is more efficient than Prasnna Kumar's algorithm [3].

The second proposed algorithm, FM (factoring method), is based on the approach of Theologou and Carlier [10] in which a one-pass method of factoring and reduction was proposed to solve the K-terminal reliability problem with imperfect nodes. The K-terminal reliability problem is to determine the probability that a specified set of nodes $K \subseteq V$ are connected, where V is the entire set of nodes in the network. In reality, the DPR problem is a logical OR-ing of *Prob*{K-terminals are connected}, as shown in Figure 1, but computing the conditional probabilities required could be rather unpleasant. Belovich [9] has proposed approximation methods in this area. Since the K-terminal reliability problem does not consider the effect of data file distribution and the set of K target nodes is not specified in a DCS, the factoring and reductions [10, 11] developed to compute the K-terminal reliability cannot be directly applied to the DPR problem. Obviously, if there are no duplicated files, i.e. if there is only one copy of each file, in the DCS, then the DPR problem can be transferred into an equivalent K-terminal reliability problem in which the K set is just the set of nodes that contain the data files needed for the programs under consideration. However, data files are usually duplicated in DCSs, so the factoring and reduction methods for the K-terminal problem cannot be directly applied to the DPR problem. General factoring and reduction methods developed for the DPR problem with perfect nodes have been proposed and discussed in [1, 2, 12]; the FM algorithm concerns the case of a DCS with imperfect nodes.

2. NOTATION AND DEFINITIONS

In this paper we will use the following notation and definitions.

2.1. Notation

$D = (V, E, F)$	An undirected DCS graph with vertex (node) set V, edge set E, and data file set F. Without loss of generality, we identify a program with a special type of data file, i.e. program $\subseteq F$.
$l_i = (e_i, u_i, v_i)$	The link i that contains the edge e_i and its two endpoints u_i and v_i.
FA_i	The set of files available at node i.
ME	A subset of E that represents the edges merged during the process of finding all MFSTs with the SM algorithm.
p_i	Reliability of node, edge or link i
q_i	$1 - p_i$
H	Subset of files of F, i.e. $H \subseteq F$, where H contains the programs to be executed and all data files needed for the execution of these programs.
$R(D_H)$	The DPR of D with a set H of needed files: Pr{all data files in H can be accessed successfully by the executed programs in H}. We omit H when no ambiguities arise.
$D - e$	The graph D with edge e deleted.
$D + e$	The graph D with edge $e = (u, v)$ contracted so that node u and v are merged into a single node. This new merged node contains all data files and programs that were in nodes u and v.

Using this notation, we can represent the DPR of program P1 that needs data files f1, f2 and f3 for its execution in the example in Figure 1 by $R(D_H)$ where $D = (V, E, F)$ and

$$
\begin{aligned}
V &= \{n_1, n_2, n_3, n_4, n_5, n_6\},\\
E &= \{e_1 = (n_1, n_2), e_2 = (n_1, n_3), e_3 = (n_2, n_3),\\
&\quad e_4 = (n_2, n_4), e_5 = (n_3, n_5), e_6 = (n_4, n_5),\\
&\quad e_7 = (n_4, n_6), n_8 = (n_5, n_6)\},\\
F &= \{\text{P1, P2, f1, f2, f3, f4, f5}\},\\
FA_{n_1} &= \{\text{f4, f5}\}, FA_{n_2} = \{\text{P1, f1}\}, FA_{n_3} = \{\text{f2}\},\\
FA_{n_4} &= \{\text{f3}\}, FA_{n_5} = \{\text{f3}\}, FA_{n_6} = \{\text{P2, f4}\}, \text{and}\\
H &= \{\text{P1, f1, f2, f3}\}.
\end{aligned}
$$

Algorithm SM
Input: the original *DCS* graph $D = (V, E, F)$ and the set H of needed files
Output: the distributed program reliability DPR
begin
repeat // reduce the original DCS graph D //
perform the degree-1 and parallel reductions
perform the series and degree-2 reduction
until no reductions can be made
Let D' be the DCS graph after the reduction step
$FOUND \leftarrow \varnothing$
$ME \leftarrow \varnothing$
FIND_FST(D', ME) // call FIND_FST to find FSTs //
for all $s, t \epsilon FOUND$ **do** // remove the FSTs which are not MFSTs //
if $t \cap s = s$ **then** remove t from $FOUND$
else if $t \cap s = t$ **then** remove s from $FOUND$ **endif**
endif
od
introduce the nodes which are endpoints of the edges in $FOUND$
apply SYREL [8] to all MFSTs stored in $FOUND$ // call the terminal reliability algorithm //
output the DPR.
end SM

FIND_FST(D, ME)
begin
if there are no FSTs in D **then return endif** // failure case //
if there exists one node n such that $FA_n \supseteq H$ **then** $FOUND \leftarrow FOUND \cup \{ME\}$
return // success case //
endif
for all $e_i \in$ the set of edges incident on the nodes containing the programs to be executed **do**
FIND_FST($D + e_i$, $ME \cup \{e_i\}$) // decompose D by Equation (3.1) //
$D \leftarrow D - e_i$
remove the irrelevant components from D
if there are no FSTs in D **then return endif**
od
end FIND_FST

ALGORITHM 1.

DEFINITION 1. *A file spanning tree (FST) [3] is a tree whose nodes hold all needed files in H.*

DEFINITION 2. *A minimal file spanning tree (MFST) [3] is a FST such that there exists no other FST that is a subset of it.*

From the definition of a MFST, the DPR can be written as

$$R(D_H) = Prob(\text{at least one MFST is operational}), \text{ or}$$

$$R(D_H) = Prob\left(\bigcup_{j=1}^{\#mfst} MFST_j\right)$$

where #mfst is the number of MFSTs for a given needed file set H.

DEFINITION 3. *A node n is called a reducible node [1] if and only if: (a) the degree of node n is two in the original DCS graph, and (b) node n is not a leaf node of any MFST.*

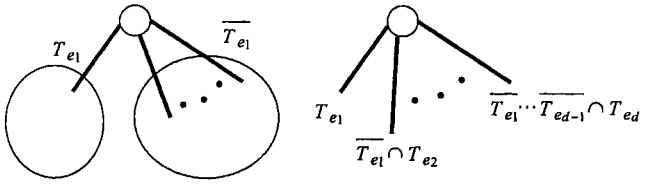

FIGURE 2. The subgraphs generated using Equation (3.1).

DEFINITION 4. *A working set is a subset of nodes of V such that if and only if all nodes in the working set fail simultaneously then there are no FSTs in the DCS.*

3. SM ALGORITHM: THE SYMBOLIC METHOD

When the DPR is computed by a symbolic method, node failure can be accounted for by using the following steps:

(i) all MFSTs are derived for the DCS with nodes considered perfect;

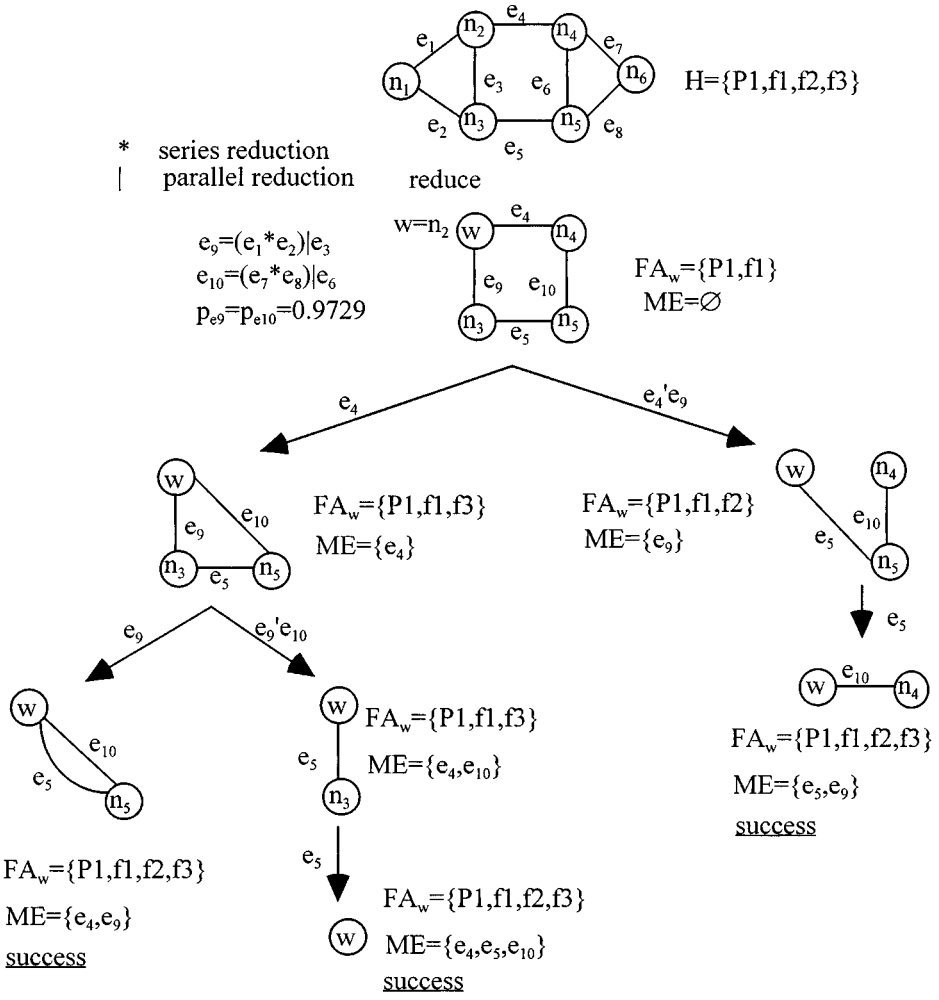

FIGURE 3. The process of finding all MFSTs using SM for the example in Figure 1.

(ii) introduce the nodes which are endpoints of the edges of MFSTs;
(iii) a terminal reliability algorithm, such as SYREL [8], is performed on the resulting expression.

To find all MFSTs, we can use the following equation proposed in FREA [1] to decompose the original DCS graph into d subgraphs:

$$\begin{aligned} R(D_H) &= p_{e_1} R(D_H + e_1) + q_{e_1} p_{e_2} R(D_H - e_1 + e_2) + \ldots \\ &\quad + q_{e_1} q_{e_2} \ldots q_{e_{d-1}} p_{e_d} R(D_H - e_1 - e_2 - \ldots \\ &\quad - e_{d-1} + e_d) \end{aligned} \tag{3.1}$$

where $\{e_1, e_2, \ldots, e_d\}$ is the set of edges incident to the nodes containing the programs being executed, and where with each subgraph we associate a set ME of edges to store the edges merged in that subgraph. This decomposition operation is performed recursively and the set ME is updated for each induced subgraph until the further induced graph is obtained in which either (a) there exists a node containing all needed data files in H, or (b) there are no FSTs. In the first of these two cases, an FST can be identified and composed by the edges stored in the set ME. After the FSTs covered by other FSTs are deleted, all MFSTs can be found.

THEOREM 3.1. *The subgraphs generated by Equation (3.1) are completely disjoint.*

Proof. According to Equation (3.1), the original graph can be decomposed into d subgraphs as shown in Figure 2. The leftmost branch corresponds to class T_{e_1}, the set of subgraphs including edge e_1. All the other branches correspond to class $\overline{T_{e_1}}$, the set of subgraphs without edges e_1. Within $\overline{T_{e_1}}$, the second branch splits it into two classes, one to include e_2, another without it, and so on. When the algorithm progresses at each internal node such kinds of splitting occur. This guarantees that no replicated subgraph will be generated. □

Since the subgraphs generated using Equation (3.1) will be completely disjoint, no duplicate ME sets will be generated during the expansion of the computation tree. Therefore, the SM does not need the CLEAN procedure used in Prasnna Kumar [3] to check and remove duplicate sets.

Before the original DCS graph is decomposed by Equation (3.1), the original DCS graph can be reduced to a

Algorithm FM
Input: the original *DCS* graph $D = (V, E, F)$ and the set H of needed files
Output: the distributed program reliability DPR
begin
  repeat // reduce the original DCS graph D //
    perform the degree-1 and parallel reductions
    perform the series and degree-2 reduction
  until no reductions can be made
  Let D' be the DCS graph after the reduction step
  Output(factoring(D'))
end FM

function factoring(D)
begin
  if there are no FSTs in D **then return** (0) **endif**
  if there exists one node n in D such that $p_n \neq 1$ (node n is an imperfect node) and
    contains the programs to be executed // identify the working set of nodes //
    then $D1 \leftarrow D$ with setting $p_n = 1$;
      $D2 \leftarrow D$ with deleting node n and its adjacent edges
      return ($p_n \cdot$ factoring($D1$) + $q_n \cdot$factoring($D2$)) // Equation (4.2) //
  endif
  return(e_factoring(D)) // call edge factoring //
end factoring

function e_factoring(D)
begin
  if there exists one node n such that $FA_n \supseteq H$ **then return** (1) **endif** // success case //
  if there are no FSTs in D **then return** (0) **endif** // failure case //
  repeat // reduction step //
    perform degree-1 and parallel reductions
    perform series and degree-2 reductions
  until no reductions can be made
  let D' be the DCS graph after the reduction step
  $R \leftarrow 0$
  $C \leftarrow 1$
  for all link $l_i = (e_i, u_i, v_i)$ in the links (edges) incident to the nodes containing
      the programs to be executed **do**
    $R \leftarrow R + C \cdot (p_{e_i} \cdot p_{u_i} \cdot p_{v_i}) \cdot$ **e_factoring**($D + e_i$) // Equation (4.3) //
    $C \leftarrow C \cdot (1 - p_{e_i} \cdot p_{u_i} \cdot p_{v_i})$
    $D \leftarrow D - e_i$
    $p'_{v_i} \leftarrow (p_{v_i} \cdot q_{e_i})/(q_{v_i} + p_{v_i} \cdot q_{e_i})$ // Equation (4.4) //
    $p'_{u_i} \leftarrow (p_{u_i} \cdot q_{e_i})/(q_{u_i} + p_{u_i} \cdot q_{e_i})$ // Equation (4.4) //
    remove the irrelevant components from D
    if there are no FSTs in D **then return** (R) **endif**
  od
  return(R)
end e_factoring

ALGORITHM 2.

smaller size by the following reduction methods developed for the DPR problem with imperfect nodes.

- *Degree-1 reduction.* A node is referred to as a degree-1 node if it has only one incident edge. Degree-1 reduction removes (i) degree-1 nodes that contain none of the needed data files and programs under consideration and (ii) their incident edges.
- *Irrelevant component deletion.* Let D' be a connected component of D that is not connected to the rest of the components of D. If there are no FSTs in D', then the component D' is irrelevant and can be deleted.
- *Parallel reduction.* Let $e_a = (u, v)$ and $e_b = (u, v)$ be two parallel edges in D. D' is obtained by replacing e_a and e_b with a single edge $e_c = (u, v)$ such that $p_{e_c} = 1 - q_{e_a} \cdot q_{e_b}$ (or $p_{e_c} = p_{e_a} + p_{e_b} - p_{e_a} \cdot p_{e_b}$).

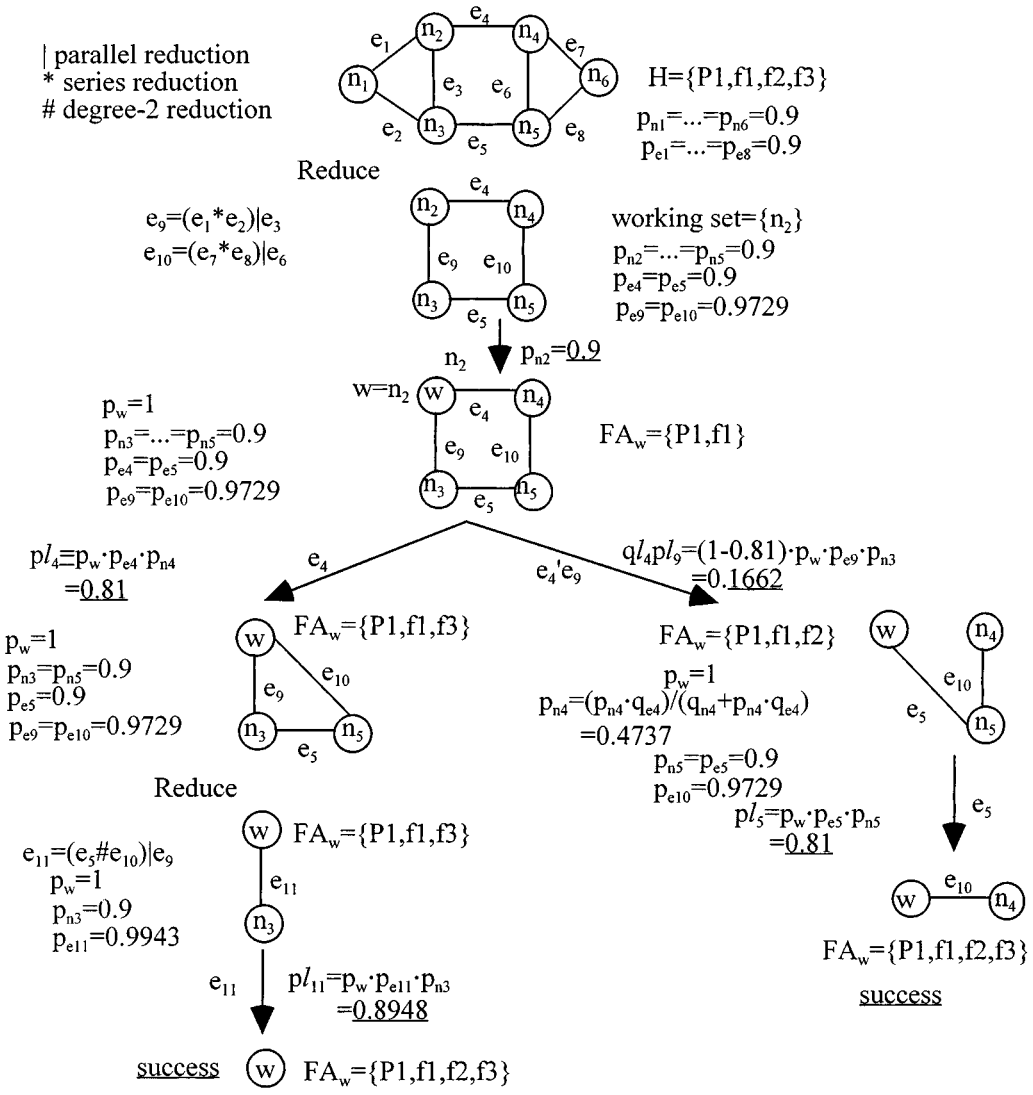

FIGURE 4. The computation tree of FM for the example in Figure 1.

The parallel reduction for the DPR problem is the same as the parallel reduction for the K-terminal network reliability problem.

- *Series reduction.* Let $e_a = (u, v)$ and $e_b = (v, w)$ be two series edges in a DCS graph D such that $degree(v) = 2$ and $FA_{v \cap H} = \varnothing$, i.e. node v contains no required data files or programs to be executed. Then a DCS graph D' is obtained by replacing e_a and e_b with a single edge $e_c = (u, w)$ such that $p_{e_c} = p_{e_a} \cdot p_v \cdot p_{e_b}$.
- *Degree-2 reduction.* Suppose node v is a reducible node; then one can apply series reduction on node v and move data files and programs within node v to a node u or w.

The complete SM algorithm can be stated as in Algorithm 1.

EXAMPLE 3.1. We use the example in Figure 1 to illustrate our symbolic method. Assume all nodes and edges have the same reliability, 0.9. The process of finding all MFSTs is shown in Figure 3.

All MFSTs found in the set *FOUND* are $\{(e_4, e_9), (e_4, e_5, e_{10}), (e_5, e_9)\}$; we then introduce the nodes which are the endpoints of the edges in *FOUND*. All MFSTs become $\{(e_4, e_9, n_2, n_3, n_4), (e_4, e_5, e_{10}, n_2, n_3, n_4, n_5), (e_5, e_9, n_2, n_3, n_5)\}$. Applying the terminal reliability algorithm SYREL [8] to these MFSTs, we obtain the disjoint terms

$$d1 = p_{n_2} \cdot p_{n_3} \cdot p_{n_4} \cdot q_{n_5} \cdot p_{e_4} \cdot p_{e_9},$$
$$d2 = p_{n_2} \cdot p_{n_3} \cdot p_{n_4} \cdot q_{n_5} \cdot p_{e_4} \cdot q_{e_5} \cdot p_{e_9},$$
$$d3 = p_{n_2} \cdot p_{n_3} \cdot p_{n_4} \cdot q_{n_5} \cdot p_{e_4} \cdot q_{e_5} \cdot p_{e_9} \cdot p_{e_{10}},$$

and

$$d4 = p_{n_2} \cdot p_{n_3} \cdot p_{n_5} \cdot p_{e_5} \cdot p_{e_9}.$$

The DPR is computed to be $\sum_{i=1}^{4} di = 0.7736$.

4. FM ALGORITHM: THE FACTORING METHOD

For the K-terminal reliability of a network with imperfect nodes, a modified factoring method has been proposed in

TABLE 1. File distributions used for comparison in the ARPA network.

	P_1	f_1	f_2	f_3	f_4	f_5
Set 1	8, 18	18, 21	9, 13	3, 18	11, 17	2, 20
Set 2	18, 21	9, 13	3, 18	11, 17	2, 20	8, 20
Set 3	5, 18	3, 8	11, 4	2, 19	8, 10	2, 19
Set 4	9, 16	15, 10	6, 8	9, 17	4, 10	3, 18
Set 5	8, 17	19, 20	8, 18	18, 21	9, 13	3, 18
Set 6	8, 15	4, 9	6, 12	20, 21	1, 12	4, 20
Set 7	11, 14	2, 18	6, 7	7, 19	3, 12	11, 16
Set 8	2, 8	2, 20	10, 5	10, 12	2, 8	7, 9
Set 9	17, 18	4, 12	4, 5	13, 15	12, 14	2, 21
Set 10	4, 9	1, 20	10, 18	15, 16	4, 16	17, 18
Set 11	2, 8, 18	6, 7, 13	5, 7, 19	3, 12, 21	1, 11, 16	11, 13, 21
Set 12	8, 9, 11	4, 10, 16	3, 5, 15	7, 11, 12	6, 8, 11	7, 10, 19
Set 13	3, 16, 21	12, 14, 19	2, 11, 13	1, 4, 14	3, 4, 17	5, 9, 13
Set 14	8, 11, 15	4, 9, 20	6, 12, 20	2, 20, 21	1, 2, 12	4, 6, 20
Set 15	5, 11, 18	5, 6, 10	2, 5, 8	12, 14, 17	5, 11, 19	4, 15, 18
Set 16	2, 5, 15	12, 15, 21	1, 13, 14	10, 16, 18	5, 17, 20	2, 4, 11
Set 17	4, 17, 19	2, 10, 18	6, 13, 14	6, 11, 12	5, 6, 20	1, 5, 8
Set 18	5, 9, 13	2, 3, 18	8, 11, 17	2, 19, 20	8, 18, 20	18, 20, 21
Set 19	11, 14, 19	4, 6, 17	10, 18, 21	2, 5, 11	6, 17, 18	5, 13, 15
Set 20	7, 12, 16	7, 11, 20	1, 17, 20	2, 3, 9	1, 7, 13	6, 18, 20
Set 21	1, 16, 18	9	2, 6, 12	5, 8, 11, 13, 17, 20	10, 14, 19, 21	3, 4, 7, 15
Set 22	1	3, 4, 9, 12, 13, 14, 17, 20	6, 7, 8, 16, 18	19	5, 10, 11, 21	2, 15
Set 23	3	5, 6, 11, 14, 15, 16, 19	8, 9, 10, 18, 20	21	7, 12, 13	1, 2, 4, 17
Set 24	7	1, 2, 9, 10, 15, 18, 19	12, 13, 14	4	3, 11, 16, 17	5, 6, 8, 21
Set 25	16, 21	3, 15	1, 10	2, 7, 8	5, 6, 9, 11, 12, 13, 17	4, 14, 18, 19, 20
Set 26	13, 18	12, 20	7	4, 5, 19	2, 3, 6, 8, 9, 10, 14, 16, 17	1, 11, 15, 21
Set 27	10	8, 12, 13	2, 21	4, 7, 15, 16	3, 5, 6, 14, 17, 18	1, 9, 11, 19, 20
Set 28	4, 11	2, 5, 6, 13, 14, 19	1, 16, 17, 18	3, 8	7, 15, 20, 21	9, 10, 12
Set 29	6, 14	2, 9	4, 7, 10, 11, 15, 19	3, 8, 18, 21	1	5, 12, 13, 16, 17, 20
Set 30	3, 4, 9, 21	7, 10	1, 11, 19	2, 6, 8, 14, 15, 16, 18	12, 13, 20	5, 17

[10]. This method is very elegant since it is a one-pass method and only requires a small additional cost for the perfect nodes case. However, the DPR problem is very different from and more complicated than the K-terminal reliability problem. Hence, the factoring and reduction methods proposed in [10] cannot be directly used for the DPR problem.

If we consider a DCS with imperfect nodes, the first step in the FM algorithm is to select a *working set* of nodes. We can choose as a working set a set of nodes including the programs to be executed. Since there is usually only one copy of each program to be executed and each program is stored in only one site in the DCS, the size of the working set we choose can be very small. Once the working set has been decided, we factor the nodes in this working set just as we do with edges. For example, if we select the working

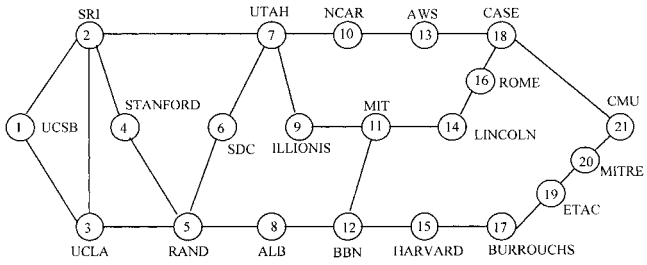

FIGURE 5. ARPA network.

TABLE 2. File distributions used for comparison in the Pacific Basin network.

	P_1	f_1	f_2	f_3	f_4
Set 1	5, 15	3, 4	12, 14	4, 6	11, 15
Set 2	3, 19	8, 13	12, 14	10, 14	13, 18
Set 3	8, 14	8, 9	6, 13	5, 6	5, 15
Set 4	10, 13	10, 15	3, 13	9, 19	8, 15
Set 5	14, 19	1, 19	8, 11	1, 7	5, 13
Set 6	9, 18	9, 10	9, 17	1, 10	10, 19
Set 7	4, 19	10, 15	2, 4	8, 18	4, 18
Set 8	13, 19	1, 8	8, 12	5, 16	1, 13
Set 9	7, 17	6, 15	4, 9	11, 14	4, 7
Set 10	4, 15	3, 10	6, 8	6, 18	3, 14
Set 11	11, 13, 19	3, 15, 17	1, 6, 14	2, 6, 18	7, 12, 14
Set 12	3, 8, 19	4, 10, 16	6, 16, 19	5, 6, 18	4, 11, 16
Set 13	7, 9, 12	11, 13, 14	1, 12, 17	10, 14, 18	4, 14, 18
Set 14	12, 15, 17	5, 7, 13	2, 8, 19	2, 3, 18	6, 9, 12
Set 15	13, 15, 16	5, 8, 13	7, 11, 12	7, 14, 15	1, 3, 4
Set 16	1, 4, 18	5, 12, 16	5, 13, 19	6, 16, 18	4, 10, 13
Set 17	7, 12, 15	6, 10, 16	5, 6, 10	1, 12, 13	7, 9, 14
Set 18	2, 8, 11	3, 4, 9	2, 11, 15	3, 16, 19	1, 7, 15
Set 19	2, 7, 9	13, 14, 19	4, 7, 13	7, 8, 17	1, 6, 15
Set 20	1, 5, 7	2, 12, 15	3, 4, 8	7, 10, 15	1, 7, 13
Set 21	6, 8, 12, 13, 16, 18	17	2, 7, 14	1, 4, 9, 11, 19	3, 5, 10, 15
Set 22	5, 10, 17, 18	3, 6, 7, 8, 11, 12	9, 16	4, 15	1, 2, 13, 14
Set 23	7, 10, 11, 18	1, 6, 9, 13	12, 16, 19	2, 4, 5, 8, 14	3, 15, 17
Set 24	2, 7, 10, 18, 19	8, 9, 12, 16	1, 3, 4, 6, 14, 17	11, 15	5, 13
Set 25	1, 5, 13, 18	6, 7, 12, 14, 15, 16, 19	2, 3, 9	10, 11	4, 8, 17
Set 26	4, 10, 12	5, 6, 11, 15	3, 9, 14, 16	13, 15, 17, 18	1, 2, 7, 8, 19
Set 27	12	6, 10, 13, 14, 16, 19	8, 9, 18	1, 2, 3, 15	4, 5, 7, 11, 17
Set 28	5	7, 8, 9, 15, 18	12, 14, 19	2, 3, 4, 10, 16	1, 6, 11, 13, 17
Set 29	14, 17, 18	2, 8, 11, 12	5, 7	1, 3, 4, 6, 16	9, 10, 13, 15, 19
Set 30	4, 10, 18	8, 11, 14, 15	6, 9, 12, 13, 17, 19	5, 7, 16	1, 2, 3

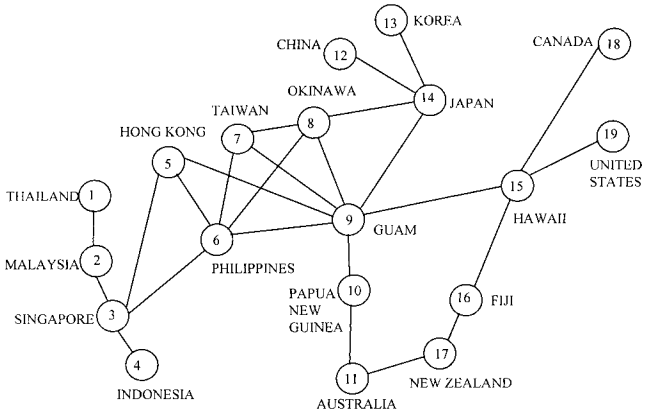

FIGURE 6. Pacific Basin network.

set $\{n_1, n_2\}$ to be factored, then there are four possible combinations in the factoring process and four different disjoint subgraphs D' will be produced from the original DCS graph D. The reliability of the original DCS can then be stated as

$$\begin{aligned} R(D_H) = {} & p_{n_1} \cdot p_{n_2} \cdot R(D_H | n_1 \text{ and } n_2 \text{ work}) \\ & + p_{n_1} \cdot q_{n_2} \cdot R(D_H | n_1 \text{ works and } n_2 \text{ failed}) \\ & + q_{n_1} \cdot p_{n_2} \cdot R(D_H | n_1 \text{ failed and } n_2 \text{ works}) \\ & + q_{n_1} \cdot q_{n_2} \cdot R(D_H | n_1 \text{ and } n_2 \text{ failed}). \end{aligned} \tag{4.1}$$

Since the set $\{n_1, n_2\}$ is a working set, by the definition of a working set, the whole DCS will fail if nodes n_1 and n_2 fail simultaneously. Hence, Equation (4.1) can be rewritten as

$$\begin{aligned} R(D_H) = {} & p_{n_1} \cdot p_{n_2} \cdot R(D_H | n_1 \text{ and } n_2 \text{ work}) \\ & + p_{n_1} \cdot q_{n_2} \cdot R(D_H | n_1 \text{ works and } n_2 \text{ failed}) \\ & + q_{n_1} \cdot p_{n_2} \cdot R(D_H | n_1 \text{ failed and } n_2 \text{ works}). \end{aligned} \tag{4.2}$$

In Equation (4.2), each subgraph of (i) D with n_1 and n_2 working, (ii) D with n_1 working and n_2 failed, and (iii)

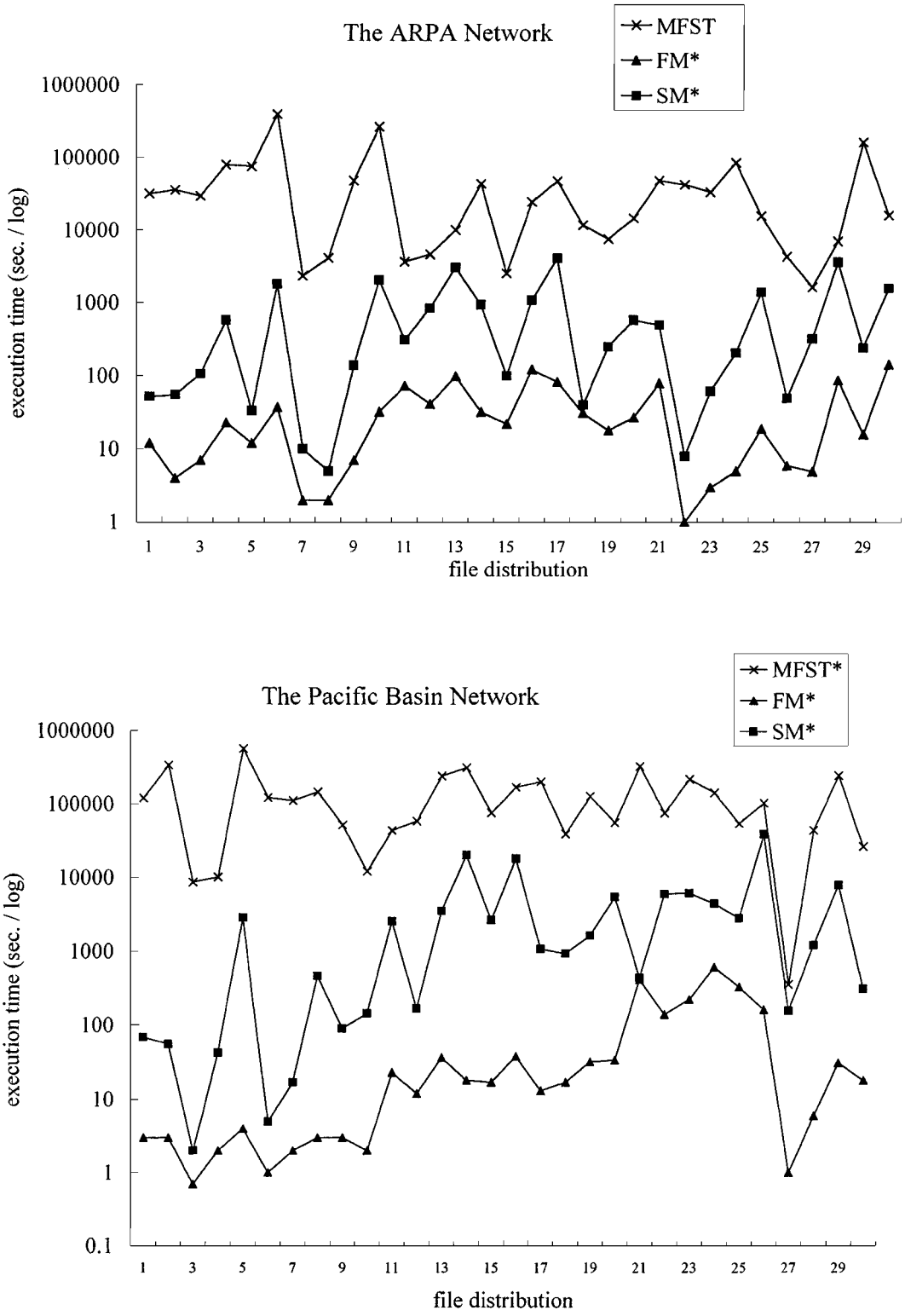

FIGURE 7. Plots of execution time for various file distributions.

D with n_1 failed and n_2 working has at least one node working, that is, either n_1 or n_2. These working nodes can be identified with perfect nodes just as the K target nodes are identified with the perfect nodes in the K-terminal problem. Once we have identified the perfect nodes in each subgraph, the factoring technique used in [10] can be applied to each subgraph D', and the technique can be generalized as follows:

$$\begin{aligned} R(D'_H) = {} & p_{l_1} \cdot R \cdot (D'_H + e_1) + q_{l_1} \dot{p}_{l_2} \\ & \cdot R(D'_H - e_1 + e_2) + \ldots + q_{l_1} \cdot q_{l_2} \ldots \\ & \cdot p_{l_d} \cdot R(D'_H - e_1 - e_2 - \ldots - e_{d-1} + e_d) \end{aligned} \tag{4.3}$$

TABLE 3. Comparison of running times (in s) for different sets of file distributions.

	The ARPA network				The Pacific Basin network			
	MFST[1]	**FM**[1]	**SM**[1]	*DPR*[1]	MFST[1]	**FM**[1]	**SM**[1]	*DPR*[1]
Set 1	31,886	12	52	0.72208	120,426	3	69	0.62321
Set 2	35,613	4	55	0.70664	336,382	3	56	0.75127
Set 3	29,709	7	106	0.85759	8704	0.7	2	0.90212
Set 4	79,298	23	576	0.77292	10,128	2	43	0.79601
Set 5	74,879	12	33	0.62265	566,992	4	2843	0.72288
Set 6	393,575	37	1797	0.62333	122,442	1	5	0.84335
Set 7	2330	2	10	0.77055	112,050	2	17	0.65175
Set 8	4141	2	5	0.84159	146,150	3	460	0.73069
Set 9	47,862	7	139	0.71358	51,973	3	90	0.86122
Set 10	263,853	32	2046	0.70700	12,201	2	144	0.86894
Set 11	3671	73	311	0.90007	44,053	23	2559	0.88738
Set 12	4652	41	845	0.90489	59,003	12	168	0.94013
Set 13	10,029	99	3040	0.87593	241,796	37	3520	0.93582
Set 14	43,565	32	936	0.89481	313,406	18	20,305	0.87653
Set 15	2538	22	100	0.85926	76,356	17	2672	0.78671
Set 16	24,679	123	1097	0.81013	169,286	38	18,246	0.89207
Set 17	47,313	83	4093	0.89348	202,565	13	1075	0.90548
Set 18	11,785	31	40	0.77215	39,632	17	934	0.93888
Set 19	7597	18	253	0.82016	128,042	32	1637	0.91192
Set 20	14,706	27	583	0.88712	56,692	34	5532	0.93303
Set 21	48,126	80	496	0.77612	327,540	414	439	0.67141
Set 22	42,324	1	8	0.48504	76,588	141	6058	0.82579
Set 23	33,687	3	62	0.53779	219,423	222	6203	0.89497
Set 24	84,666	5	208	0.70892	142,629	608	4479	0.81395
Set 25	15,848	19	1414	0.73106	55,135	330	2831	0.79374
Set 26	4382	6	50	0.67767	104,456	162	39681	0.92176
Set 27	1663	5	329	0.76526	356	1	156	0.67407
Set 28	7130	88	3666	0.88650	44,681	6	1206	0.81330
Set 29	161,932	16	246	0.76793	246,909	31	8060	0.91986
Set 30	16,200	146	1599	0.83323	26,970	18	311	0.81735

[1]A Sun SPARC system 600 workstation was used to run the program(s).

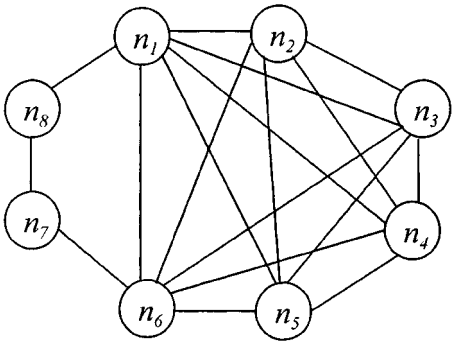

FIGURE 8. Benchmark DCS $D_{8,6}$.

TABLE 4. File distribution table.

FA_{n1}	= {f1, f2, f3}
FA_{n2}	= {f2, f3, f4}
FA_{n3}	= {f3, f4, f5}
FA_{n4}	= {f4, f5, f6}
FA_{n5}	= {f5, f6 ,f7}
FA_{n6}	= {f6, f7, f8}
FA_{n7}	= {f1, f7, f8}
FA_{n8}	= {f1, f2, f8}
FA_{n9}	= {f3, f7, f8}
FA_{n10}	= {f1, f4, f7}

where D' is a subgraph generated by the process of factoring on the working set of D, the set $\{l_1 = (e_1, v_1, u_1), l_2 = (e_2, v_2, u_2), \ldots, l_d = (e_d, v_d, u_d)\}$ is the set of links incident to the nodes that are identified with perfect nodes and $p_{l_i} = p_{e_i} \cdot p_{v_i} \cdot p_{u_i}$ is the probability of link i working, for $i = 1$ to d. In each subgraph of D' with edge $e_i = (u_i, v_i)$ deleted, the reliability of the two endpoints u_i and v_i, is updated with the new reliability

$$p'_{v_i} = (p_{v_i} \cdot q_{e_i})/(q_{v_i} + p_{v_i} \cdot q_{e_i}), \text{ and}$$
$$p'_{u_i} = (p_{u_i} \cdot q_{e_i})/(q_{u_i} + p_{u_i} \cdot q_{e_i}) \quad \text{(see [10])}. \qquad (4.4)$$

TABLE 5. Number of subgraphs generated and actual execution time against different topologies.

	Number of subgraphs generated				Execution time (s)				
DCS	MFST[1]	ENR/KW[2]	**SM**[1]	**FM**[1]	MFST[1]	ENR/KW[2]	**SM**[1]	**FM**[1]	DPR[1]
$D_{8,4}$	35	16	10	8	<0.01	<0.01	<0.01	<0.01	0.8915513
$D_{10,4}$	55	20	9	7	< 0.01	0.01	<0.01	<0.01	0.8893553
$D_{8,6}$	306	72	32	20	0.02	0.02	0.01	<0.01	0.8988961
$D_{8,7}$	1159	289	181	64	0.81	0.06	0.28	0.01	0.8990611
$D_{10,7}$	3443	462	151	60	1.30	0.15	0.21	0.02	0.8990729
$D_{8,8}$	3225	1196	1356	260	21.83	0.22	20.66	0.08	0.8990899
$D_{10,8}$	20,464	2556	1195	250	103.82	0.82	16.69	0.09	0.8990949
$D_{10,9}$	131,899	17,832	8619	1255	5023.68	5.64	183.72	0.51	0.8990990

[1]An Intel Pentium P133 PC was used to run the program(s).
[2]These values were obtained by Ke and Wang [5].

Equations (4.3) and (4.4) and the reduction methods described in Section 3 can be recursively applied to each induced graph until either (i) the further induced graph with a node contains all needed data files and all programs to be executed or (ii) the further induced graph contains no FSTs. The former case represents success (reliability $= 1$); the latter case represents failure (reliability $= 0$). The complete FM algorithm is stated in Algorithm 2.

EXAMPLE 4.1. To illustrate the FM algorithm, we shall again use the example in Figure 1. Assume the reliability of all the nodes and links is 0.9. The complete computation tree of FM is shown in Figure 4.

From Figure 4, the DPR can be computed as

$$\text{DPR} = 0.9 \cdot 0.81 \cdot 0.8948 + 0.9 \cdot 0.1662 \cdot 0.81 = 0.7736.$$

5. COMPLEXITY AND COMPARISONS

5.1. Complexity analysis

It is well known that computing K-terminal reliability in general is NP-hard, or $\#P$-complete [13]. However, for some classes of networks, for example, tree and series-parallel networks, the K-terminal reliability problems can be computed in linear time by applying well known reductions like series, degree-2, parallel and polygon-to-chain reductions [14]. However, the DPR problem is much more complicated than the K-terminal problem, since its computational complexity is dependent not only on the topology of the network but also on the file distributions. Actually we have shown that the DPR problem for series-parallel, tree and star networks is still NP-hard [15]. Therefore, there exists no polynomial time algorithm to compute the reliability of the distributed program for general distributed computing systems. Naturally, the SM and FM require exponential time, i.e. $2^{|V|+|E|}$, in the worst case. An appropriate and rational comparison for these different algorithms can be made based on the counting approach which counts the number of intermediate trees or subgraphs generated during the whole reliability evaluation. From such a comparison, one can tell how much memory space and time units are required for their algorithms to run the distributed programs under the effects of different topologies of the DCS and file distributions.

5.2. Comparisons

The algorithms developed in this paper will now be compared with the existing algorithms under the changes in:

(i) the file distribution on the nodes of the DCS; and
(ii) the topology of DCS.

5.2.1. *Effect of data file distributions on performance of different algorithms*

The algorithms were tested using the examples of the ARPA network and the Pacific Basin network. The topologies of the ARPA and Pacific Basin networks are shown in Figures 5 and 6. All edges and nodes have reliability $= 0.9$. Thirty sets of file distributions, generated randomly, for each of the ARPA and Pacific Basic networks are listed in Tables 1 and 2. In Tables 1 and 2, sets, 1–10 are those file distributions for which two copies of each file are distributed randomly in the network while sets 11–20 are for three copies of each file. We also generate randomly the other kind of file distribution, sets 21–30, where each node contains only one data file. Table 3 gives running times and the reliability obtained for the example of Tables 1 and 2. The execution time is also plotted in Figure 7. From Table 3 and Figure 7, it is clear that the SM and FM algorithms are much more efficient than Prasnna Kumar's [3] algorithm.

5.2.2. *Effect of topology on performance of different algorithms*

In this study, we want to see the effect of topological configuration on the performance of different algorithms used. Consider the benchmark networks given in ENR/KW [5]. Let D_{ij} be the benchmark DCS with i nodes and node n_1 to node n_j being completely connected. Figure 8 depicts the example of $D_{8,6}$. The file distributions are given in Table 4. Assume program P1 needs data files $\{f1, f3, f5\}$

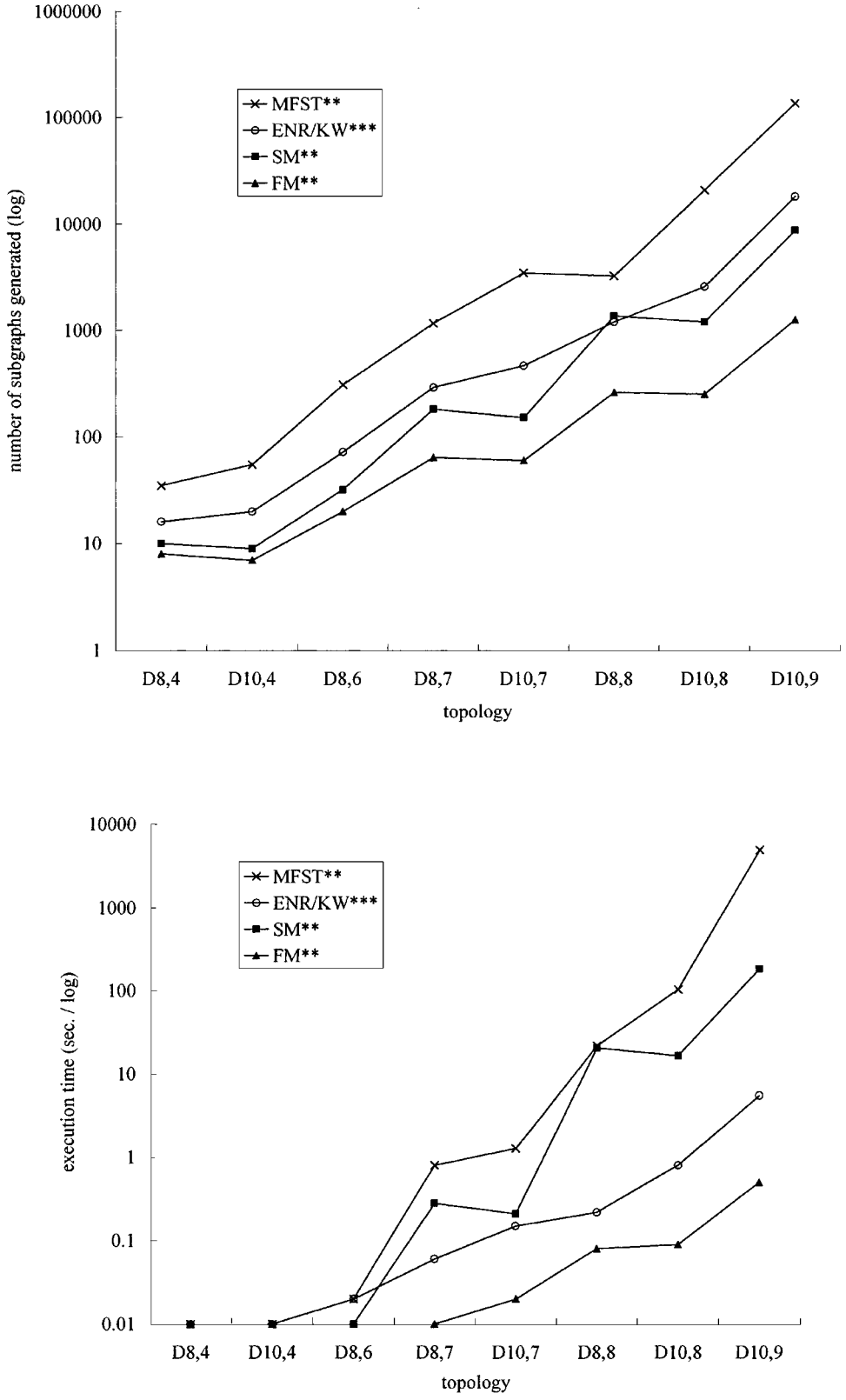

FIGURE 9. Plots of the number of subgraphs generated and execution time for topology variations.

for its executions. These topologies and file distributions are the same as those used in ENR/WK [5]. Table 5 shows the number of subgraphs generated and the actual execution time against different topologies based on program P1 as executed at node n_1. In addition, Figure 9 shows the plots of Table 5. From Table 5 and Figure 9, it is clear that the FM algorithm is much more efficient, compared with the other algorithms, in any of these different topologies.

6. CONCLUSION

In this paper we have proposed two algorithms for computing the reliability of distributed computing systems with imperfect nodes. The first algorithm, SM, is a two-pass method that requires the terminal reliability algorithms to compute the reliability values of symbolic expressions obtained from the SM. The second algorithm, the FM, uses a factoring approach that directly computes the reliability without enumerating the symbolic expressions. In addition, the use of various reliability preserving reduction techniques in the SM and FM implies that the size of the graph will be reduced and, therefore, fewer subgraphs will be generated. Comparisons with existing methods on various file distributions and network topologies show the usefulness of the FM algorithm for complex DCSs.

REFERENCES

[1] Chen, D. J. and Lin, M. S. (1994) On distributed computing systems reliability analysis under program execution constraints. *IEEE Trans. Comput.*, **16**, 87–97.

[2] Chen, D. J. and Huang, T. H. (1992) Reliability analysis of distributed systems based on a fast reliability algorithm. *IEEE Trans. Parallel Distrib. Syst.*, **3**, 139–153.

[3] Prasnna Kumar, V. K., Hariri, S. and Raghavendra, C. S. (1986) Distributed program reliability analysis. *IEEE Trans. Software Eng.*, **12**, 42–50.

[4] Kumar, A., Rai, S. and Agrawal, D. P. (1988) On computer communication network reliability under program execution constraints. *IEEE J. Select. Areas Commun.*, **6**, 1393–1399.

[5] Ke, W. J. and Wang, S. D. (1997) Reliability evaluation for distributed computing networks with imperfect nodes. *IEEE Trans. Reliability*, **46**, 342–349.

[6] Lopez-Benitez, N. (1994) Dependability modeling and analysis of distributed programs. *IEEE Trans. Software Eng.*, **20**, 345–352.

[7] Kumar, A. and Agrawal, D. P. (1993) A generalized algorithm for evaluating distributed program reliability. *IEEE Trans. Reliability*, **42**, 416–426.

[8] Hariri, S. and Raghavendra, C. S. (1987) *SYREL: A Symbolic Reliability Algorithm based on Path and Cutset Methods*. *IEEE Trans. Computers*, **36**, 1224–1232.

[9] Belovich, S. G. (1995) A design technique for reliable networks under a non-uniform traffic distribution. *IEEE Trans. Reliability*, **44**, 377–387.

[10] Theologou, O. R. and Carlier, J. G. (1991) Factoring and reductions for networks with imperfect vertices. *IEEE Trans. Reliability*, **40**, 210–217.

[11] Satyanarayana, A., Chang, M. K. (1983) Network reliability and the factoring theorem. *Networks*, **13**, 107–120.

[12] Lin, M. S. and Chen, D. J. (1993) General reduction methods for the reliability analysis of distributed computing systems. *Comp. J.*, **36**, 631–644.

[13] Ball, M. O. (1986) Computational complexity of network reliability analysis: an overview. *IEEE Trans. Reliability*, **35**, 230–239.

[14] Satyanarayana, A. and Wood, R. K. (1985) A linear-time algorithm for computing K-terminal reliability in series-parallel networks. *SIAM J. Comput.*, **14**, 818–832.

[15] Lin, M. S. and Chen, D. J. (1997) The computational complexity of the reliability problem on distributed systems. *Inform. Process. Lett.*, **64**, 143–147.