Pergamon

Computers Math. Applic. Vol. 34, No. 10, pp. 115–131, 1997

0898-1221/97 $17.00 + 0.00

PII: S0898-1221(97)00210-1

A Heuristic Approach to Generating File Spanning Trees for Reliability Analysis of Distributed Computing Systems*

DENG-JYI CHEN, RUEY-SHUN CHEN** AND TIEN-HSIANG HUANG
Institute of Computer Science and Information Engineering
National Chiao Tung University, Hsinchu, Taiwan, R.O.C.

(Received and accepted October 1994)

Abstract—The reliability of Distributed Computing Systems (DCS) in terms of Distributed Program Reliability (DPR) and Distributed System Reliability (DSR) has been studied intensively. Current reliability algorithms available for the analysis of DPR and DSR include MFST, FARE, FST, and FST-SPR. This paper presents a reliability algorithm, called HRFST, that eliminates the need to search a spanning tree during each subgraph generation. The HRFST algorithm reduces both the number of subgraphs (or trees) generated and the actual execution time required for analysis of DPR and DSR. Examination of several sample cases shows that the HRFST algorithm is more efficient than the FST-SPR algorithm.

Keywords—Distributed Computing Systems (DCS), Distributed Program Reliability (DPR), Distributed System Reliability (DSR), Reliability.

1. INTRODUCTION

In reliability analysis of Distributed Computing Systems (DCS), V.K.P. Kumar has proposed a very useful notion called a Minimal File Spanning Tree (MFST) and developed an algorithm called MFST to find MFSTs within a graph [1,2]. The MFST algorithm takes two passes to obtain the reliability of PDS. Pass 1 is to obtain the multiterminal connections for every MFST. Pass 2 is to use an algorithm called SYREL [3] to compute the equivalent reliability expressions by the multiterminal connections of every MFST. To improve the MFST algorithm, A. Kumar developed an algorithm called FARE (Fast Algorithm for Reliability Evaluation) [4,5] to compute the DPR and DSR. The FARE algorithm uses a connection matrix to represent each MFST and proposes some simplified techniques for speeding up the analysis process. To further improve the evaluation speed, we also proposed the FST-SPR algorithm for reducing the number of subgraphs generated during reliability evaluation [6]. In [6], the FST-SPR algorithm was compared with the MFST and FARE algorithms to show its performance advantage. The basic idea behind the FST-SPR algorithm is to make subgraphs generated completely disjoint, so that no replicated subgraphs are generated during the reliability evaluation process. In order to generate disjoint subgraphs, we have to search a spanning tree from the current graph and then cut each edge in the spanning tree disjointly to produce the disjoint subgraphs. This process is repeatedly applied to each subgraph generated until a File Spanning Tree (FST) is found or no edge is reached.

*This research work was supported in part by the National Science Council of the R.O.C. and in part by the Chung San Institute of Technology, Taiwan, R.O.C.
**Author to whom all correspondence should be addressed.

Therefore, to find a spanning tree from each generated subgraph can be computationally costly and time consuming.

In this paper, we present a reliability algorithm called HRFST that eliminates the need to search of a spanning tree during the generation of each subgraph. The HRFST algorithm reduces both the number of subgraphs (or trees) generated and the actual execution time required for the analysis of DPR and DSR. Examination of various sample cases clearly shows that the HRFST is more efficient than the FST-SPR algorithm.

2. PREVIOUS WORK

The notation and definitions used in [7] are recalled here for consistency.

2.1. Notation

x_i: a node representing a processing element i

$x_{i,j}$: a link between processing elements i and j

$p_{i,j}(q_{i,j})$: probability that the link $x_{i,j}$ is working (failure)

t: a subgraph, which can be a tree or forests of the DPS graph (the trees and forests are represented by sets of nodes and links)

Fi: the data file i

Pi: the distributed program i

FA_i: the set of data files available at processing element x_i

FA_t: the set of data files available by subgraph t

PA_i: the set of programs available at processing element x_i

PA_t: the set of programs available by subgraph t

PN: the set of programs that need to be executed in the DPS

FN_i: the set of data files needed for program i to be executed

FN: the set of data files needed for several programs to be executed in the DPS

LS_t: a set of links that represents the links' state in the subgraph t; the state of each link in the set is:

failure	if $\overline{x_{i,j}} \in LS_t$
working	if $x_{i,j} \in LS_t$
do not care	otherwise

ST_t: a set of links that can be used to represent a spanning tree of the subgraph t

PR_t: a probability expression denoted by $p_{i,j}$ or $q_{i,j}$ that represents the probability of the subgraph t containing a working spanning tree

R_t: the reliability of the subgraph t

R: the reliability of the DPS

2.2. Definitions

DEFINITION 1. An **FST** is an File Spanning Tree that connects the root node (the processing element that runs the program under consideration) to some other nodes such that its vertices hold all the necessary data files for the programs to be executed.

DEFINITION 2. An **MFST** is a minimal FST such that there exists no other FST which is a subset of it.

DEFINITION 3. A **probability space** is composed by all possible states of the links (working, failure, or do not care).

DEFINITION 4. A **probability graph** is a graph that has a probability space associated with it. For the original graph, the probability is assumed to be 1 (all links do not care). Also, the probability space of a subgraph that will be equal to the sum of the probability space of all subgraphs generated by this subgraph.

DEFINITION 5. A subgraph t is **satisfied** means that the data files and programs available by subgraph t can execute the distributed program successfully. ($FA_t \supseteq FN$, and $PA_t \supseteq PN$, and the failure links will not affect the program's execution.)

DEFINITION 6. A subgraph t may generate several subgraphs by cutting its links. Then the subgraph t is called a parent subgraph and the subgraphs generated from subgraph t are called child subgraphs of subgraph t.

2.3. The FST-SPR Algorithm

Consider the distributed processing system in Figure 1, there are four processing elements (x_1, x_2, x_3, x_4) connected by links $x_{1,2}$, $x_{1,3}$, $x_{2,3}$, $x_{2,4}$, and $x_{3,4}$. Processing element x_1 contains two data files ($F1$ and $F2$) and can run program 1 directly to communicate with other nodes to access data files required to complete the execution of program 1. Detailed information on each node is summarized in FA_j and PA_j $(j = 1, 2, \ldots, 4)$.

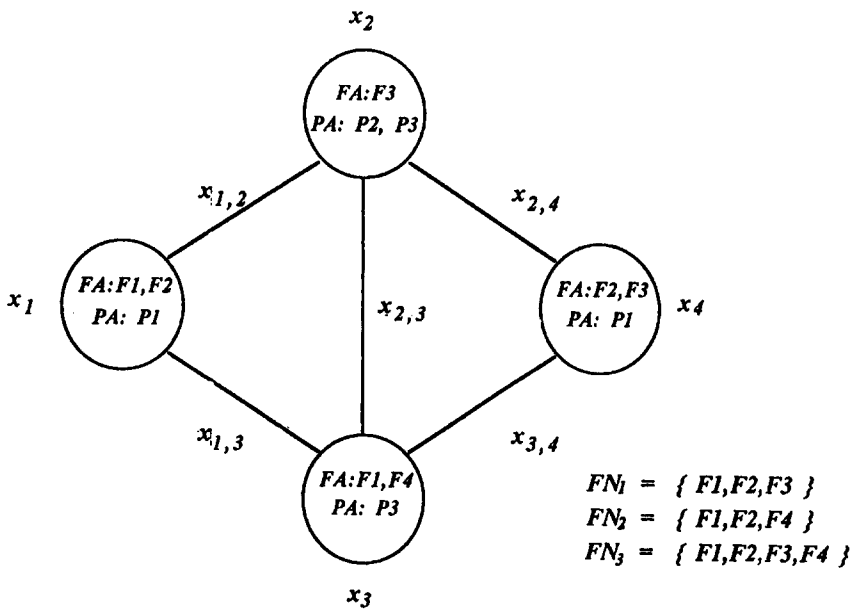

Figure 1. A simple DPS with four processing elements.

An outline of the FST-SPR algorithm used to compute the DPR and DSR presented here. For a more detailed treatment, readers are referred to [6].

Step 1. Perform reliability-preserving reduction on the original DPS graph.

Step 2. Find a spanning tree from the reduced DPS graph.

Step 3. Cut each link from the spanning tree (obtained from Step 1) so that the resulting subgraphs are all completely disjoint.

Step 4. Check whether each the resulting subgraph contains an FST. If so, then repeat Steps 1 to 4 until the resulting subgraph contains no FST or the resulting subgraph contains no link.

Step 5. For all the subgraphs generated during the cutting process, sum all the probability subgraphs that contain an FST by adding all the associated probability spaces to obtain the final reliability.

Subgraphs generated using the FST-SPR reliability algorithm are completely disjoint, and hence, no replicated subgraphs (or trees) will be generated. In [6], the FST-SPR algorithm was compared with MFST [1] and FARE [5] to show its performance advantage in the analysis of DPR and DSR.

3. THE HRFST ALGORITHM

3.1. Observations on the FST-SPR Algorithm

When we study the FST-SPR algorithm carefully, we find that a spanning tree must be found each time for a subgraph is generated (as shown in Step 2 of the FST-SPR algorithm). This implies that the number of times the spanning tree generation procedure is invoked is equal to the number of subgraphs generated. In the worst case, the computation cost will be $L!$, where L is the number of links in the first spanning tree identified in the original graph. Thus, the cost to find a spanning tree in the FST-SPR algorithm is high.

The purpose of finding a spanning tree is to see if the current subgraph can run the distributed program successfully (whether it contains all the data files required for the execution of the distributed program under consideration). If it can, then a disjoint-cutting process is performed to generate disjoint subgraphs from the current subgraph. If it cannot, then the subgraph generation is stopped. This implies that a spanning tree that can run the distributed program successfully will also be an FST and that it will take more time, in general, to run the distributed program, since the number of nodes and links of a spanning tree is greater than the number of nodes and links of an FST. Thus, using the spanning tree to run the distributed program will take more time and generate more subgraphs than that of the FST during the analysis of DPR and DSR.

Based on the above observations, we suggest that if we can find a way to choose an FST together with an appropriate cutting approach to generating subgraphs, then both the computation time and the number of subgraphs can be reduced. This suggestion can be justified easily. Consider the example in Figure 2, and suppose links $L_1, L_2, \ldots, L_5$ in graph A are a spanning tree. Then five disjoint subgraphs will be generated from graph A by applying the FST-SPR algorithm. Suppose links L_1, L_2, L_3 in the same spanning tree are an FST for a distributed program Pj. Then the operation of links L_1, L_2, L_3 in graph A will be enough for successful execution of program Pj. Thus, finding the FST instead of the spanning tree for the disjoint-cutting process will generate fewer subgraphs. The difference between the use of an FST and a spanning tree for subgraph generation is shown in Figure 2.

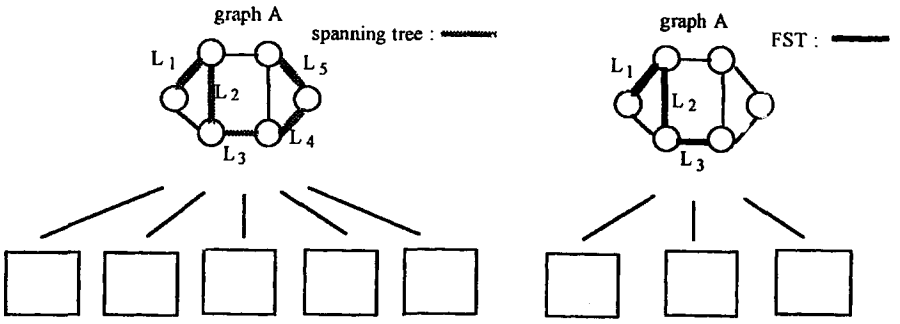

Figure 2. Difference between the use of spanning tree and FST.

3.2. The Conceptual Foundation of the HRFST Algorithm

Once the motivation is understood, we need to construct a method of finding an FST with reasonable cost and then to apply the disjoint-cutting process to generate subgraphs. The most straightforward approach is to use the MFST algorithm to find an FST. This is not a good solution, however, since we have been trying to avoid generating replicated trees (subgraphs), in order to reduce the computation time. Thus, a new approach must be developed.

Here we will present a new method of finding an FST. The basic idea of the method is to find a heuristic cost function to compute the cost of each link $x_{i,j}$ in terms of data files and programs resident in nodes x_i and x_j. Through the cost function analysis, we will be able to understand,

which connecting between nodes will offer us a good chance of obtaining an FST. The heuristic cost function is defined below.

$$\text{cost}(x_{i,j}) = \#(FN(FA_i \cup FA_j)) + \#(PN - (PA_i \cup PA_j)),$$

where "–" is the difference set and "#" is the number of sets.

Therefore, $\text{cost}(x_{ij}) = 0$ means that if link $x_{i,j}$ is selected then there is a good chance that an FST is in subgraph $x_j, x_{i,j}, x_j$. Rules for selecting links are listed below.

Rule 1. Always choose the link that has the minimum cost in the graph.

Rule 2. If there are two or more links with the minimum cost, then choose the one whose connecting nodes have the least replicated data files and programs among those under consideration.

Rule 1 indicates that we should use the most important link as the factor for the probability partition process; Rule 2 tells us that connecting two nodes that have the least-replicated data files and programs will decrease the costs of the adjacent links. This will enable an FST to be found as soon as possible.

Consider the DPS in Figure 1 for the analysis of DPR1. The costs of links $x_{1,2}$, $x_{1,3}$, $x_{2,3}$, $x_{2,4}$, and $x_{3,4}$ are, based on the cost function defined above, 0, 1, 2, 1, and 0, respectively. According to the link selecting rule, link $x_{1,2}$ is selected to be cut for the generation of subgraphs. Since its cost is zero, there is only one subgraph generated by the original graph. The probability space after this partition is shown in Figure 3.

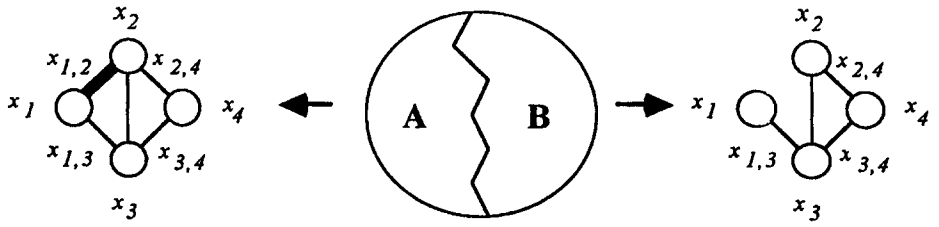

Figure 3. The probability space of the original graph.

If we continue the analysis process, the costs of links $x_{1,3}$, $x_{2,3}$, $x_{2,4}$, and $x_{3,4}$ are now 1, 2, 1, and 0, respectively. Since the link $x_{3,4}$ has the minimum cost (zero), the subgraphs of portion B are the subgraph B without link $x_{3,4}$ (shown by portion $B2$). The probability space after this partition is shown in Figure 4.

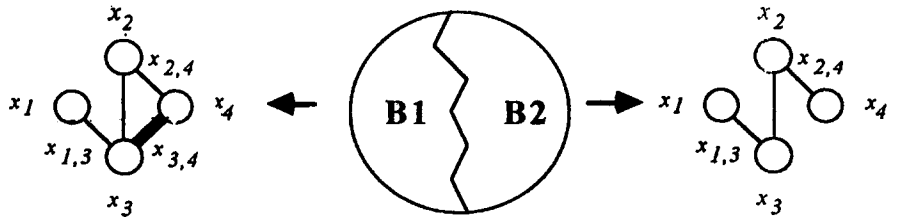

Figure 4. The probability space of the portion B.

The portion $B2$ now has to be split. The costs of links $x_{1,3}$, $x_{2,3}$, and $x_{2,4}$ are 1, 2, and 1, respectively, so the first link to be cut is $x_{1,3}$. Since the cost of link $x_{1,3}$ is not zero, another link should be selected to be cut for generating the second subgraph. Because of the disjoint property, the second subgraph of subgraph $B2$ should merge the nodes x_1 and x_3, and now the costs of links $x_{2,3}$ and $x_{2,4}$ are 0 and 1. Since the cost of link $x_{2,3}$ is the minimum and is zero, then the

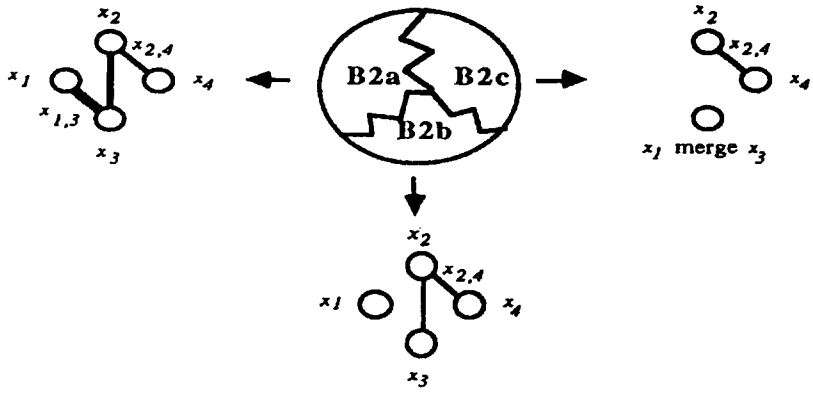

Figure 5. The probability space of the portion $B2$.

link to be cut in the second subgraph is $x_{2,3}$. The probability space after this partition is shown in Figure 5.

The PR_t of every subgraph can be computed according to the LS_t and a set called WORK_t, which is the set of the links to be cut by the subgraphs (multiply $p_{i,j}$ if $x_{i,j} \in LS_t$ or $x_{i,j} \in \text{WORK}_t$, multiply $q_{i,j}$ if $\overline{x_{i,j}} \in LS_t$). Finally, we sum all the PR_t's to obtain the reliability of the DPS.

3.2. The Complete HRFST Reliability Algorithm

The HRFST algorithm is different from the FST-SPR algorithm; it is described informally below.

1. Perform as many reductions on the current graph as possible and compute the cost of each link while carrying out the reductions.
2. If there is any single node that contains all data files and programs required, then remove current graph from TRY, store the current graph in the list FOUND, and go to Step 5.
3. Remove the current graph from TRY. If it has no links that can be cut, then store the current graph in the list FOUND and go to Step 7.
4. Use Rule 1 and Rule 2 to select a link $x_{i,j}$. Cut the selected link $x_{i,j}$ from the current graph to generate the first subgraph. Connect nodes x_i and x_j and perform reductions if necessary. Repeat the same process to get a link $x_{k,l}$ and cut link $x_{k,l}$ to generate the second subgraph, until a link $x_{m,n}$ with cost zero is chosen to produce the n^{th} child subgraph.
5. Set the links chosen in Step 4 to in the current graph. This implies that the current graph contains at least one FST. Add the current graph and the working links to FOUND.
6. Check each subgraph generated in Step 4 and add the candidate FST subgraphs to the list TRY.
7. For each subgraph in TRY, apply Steps 1 to 6 repeatedly until list TRY is empty.
8. All the FSTs of each size are now stored in the list FOUND.

The formal algorithm is presented below.

HRFST ALGORITHM.

begin

step 1: initialization

t = original graph ;

TRY = $\{t\}$;

FOUND = f ;

LS_t = f ;

$FN = FNi \cup FN_i$ (where program $i \in PN$) ;

$R = 0$;
step 2 : compute_cost(t) ;
step 3 : generate subgraphs
repeat
3.1.1 get a graph t from TRY and remove t from TRY ;
3.1.2 reduction step
repeat
degree-1_reduction(t) ;
degree-2_reduction(t);
series_reduction(t) ;
parallel_reduction(t) ;
unsatisfied_connected_components_deletion(t) ;
until no reduction happen
3.2 checking step
if there is any single node i where $FA_i \supseteq FN$ and $PA_i \supseteq PN$
then set PR_t according to LS_t add t to FOUND; continue;
find any connected component i in t such that
$FA_i \supseteq FN$ and $PA_i \subseteq PN$;
if any connected component i exists
then add t to FOUND ;
else continue ;
3.3 cutting step
add subgraph (t, WORK_t) to TRY ;
for all link $x_{i,j} \in \text{WORK}_t$;
$PR_t = PR_t * p_{i,j}$;
od
until (TRY = f)
step 4 : compute reliability
for all $t \in$ FOUND
$R = R + PR_t$;
od

end

procedure subgraph (t, WORK_t)
begin
$\text{WORK}_t = f$
repeat
$child \leftarrow t$;
$PR_{child} = PR_t$;
$LS_{child} = LS_t$;
$child \leftarrow$ nodes_merged($child$, WORK_t) ;
for all $x_{i,j} \in \text{WORK}_t$ **do**
$PR_{child} = PR_{child} * p_{i,j}$;
od
$x_{i,j} \leftarrow$ min_cost_link($child$) ;
$child \leftarrow$ cut_link($child, x_{i,j}$) ;
$CUT_{child} = \{x_{i,j}\}$;
$PR_{child} = PR_{child} * q_{i,j}$;
$LS_{child} = LS_{child} \gg \{\overline{x_{i,j}}\}$
$\text{WORK}_t = \text{WORK}_t \gg \{x_{i,j}\}$;
until (cost($x_{i,j}$) = 0)

return(*child*)

end

procedure compute_cost(t)
begin

for each link $x_{i,j}$ in subgraph t **do**
$\text{cost}(x_{i,j}) = \#(FN - (FA_i \cup FA_j)) + \#(PN - (PA_i \cup PA_j))$
od

end

procedure modify_cost($t, x_{i,j}$)
begin

$\text{cost}(x_{i,j}) = \#(FN - (FA_i \cup FA_j)) + \#(PN - (PA_i \cup PA_j))$

end

procedure min_cost_link(t) ;
begin

select a link $x_{i,j}$ whose cost is minimum from all the links in subgraph t ;
if there are two or more links with the same minimum cost, then select the one whose connecting nodes have the least-replicated data files and programs.

end

procedure series_reduction(t)
begin

for all node x_i in t
/* **similar to that in the FST-SPR algorithm** */
modify_cost($t, x_{k,j}$)
od

end

procedure parallel_ reduction(t)
begin

for all node x_i in t
/* **similar to that in the FST-SPR algorithm** */
modify_cost($t, x_{k,j}$)
od

end

procedure degree-1_reduction(t)
begin

/* **similar to that in the FST-SPR algorithm** */
modify_cost($t, x_{k,j}$)

end

procedure degree-2_reduction(t)
begin

for all node x_i in t
/* **similar to that in the FST-SPR algorithm** */
modify_cost($t, x_{k,j}$)
od

end

procedure nodes_merged(t,old)
begin
  for all link $x_{i,j} \in$ old
    /* the same as that in the HRFST algorithm */
    for all nodes x_k which is a neighbor node of x_i
      modify_cost($t, x_{k,i}$)
    od
  od
end

procedure unsatisfied_connected_components_deletion (t)
begin
  for all unsatisfied connected components i in graph t
    delete all nodes x_n in connected component i ;
    delete all links $x_{i,j}$ in connected component i ;
  od
end

4. RELIABILITY ANALYSIS OF DPS USING THE HFRST RELIABILITY ALGORITHMS

4.1. Example

We shall now apply the HRFST algorithm to the DPS in Figure 1 to analyze the DPR1. Splitting snapshots of the subgraphs generated are illustrated in Figure 6.

$$\text{DPR1} = p_{1,2} + q_{1,2}\,p_{3,4} + q_{1,2}\,p_{1,3}\,p_{2,3}\,q_{3,4} + q_{1,2}\,q_{1,3}\,p_{2,3}\,p_{2,4}\,q_{3,4}.$$

If we assume all the links have the same reliability, 0.9, then DPR1 is equal to 0.99891. To evaluate the DSR, $FN = \{F1, F2, F3, F4\}$. Applying the same algorithm, we obtain

$$\begin{aligned}\text{DSR} = {} & p_{1,2}\,p_{2,3} + p_{1,2}\,p_{1,3}\,q_{2,3} + p_{1,2}\,q_{1,3}\,q_{2,3}\,p_{2,4}\,p_{3,4} + q_{1,2}\,p_{1,3}\,p_{2,3} \\ & + q_{1,2}\,q_{2,3}\,p_{2,4}\,p_{3,4} + q_{1,2}\,q_{1,3}\,p_{2,3}\,q_{2,4}\,p_{3,4} + q_{1,2}\,q_{1,3}\,p_{2,3}\,p_{2,4}.\end{aligned}$$

Again, if we assume all the links have the same reliability, 0.9, then the DSR is equal to 0.9963. The results obtained by the HRFST algorithm are equivalent to those obtained with the FST-SPR algorithm.

4.2. The Correctness and Time Complexity of the Algorithm

THEOREM 1. *The FST-SPR algorithm is correct.*

PROOF. See [7].

THEOREM 2. *The HRFST algorithm is correct.*

PROOF. The cutting method of the HRFST algorithm is also based on the factoring theorem and is equivalent to the form of the equation

$$\begin{aligned}R(G) = {} & p_{i,j}, p_{k,l} \ldots p_{y,z} p_{u,v} R(G_n) + p_{i,j} p_{k,l} \ldots p_{y,z} q_{u,v} R(G'_n) \\ & + p_{i,j} p_{k,l} \ldots q_{y,z} R(G'_{n-1}) + \cdots + q_{i,j} R(G'_1).\end{aligned}$$

Further, the topology meaning of the factors is an FST instead of a spanning tree. Thus, the disjoint property still holds in the HRFST algorithm. All the reduction techniques are also

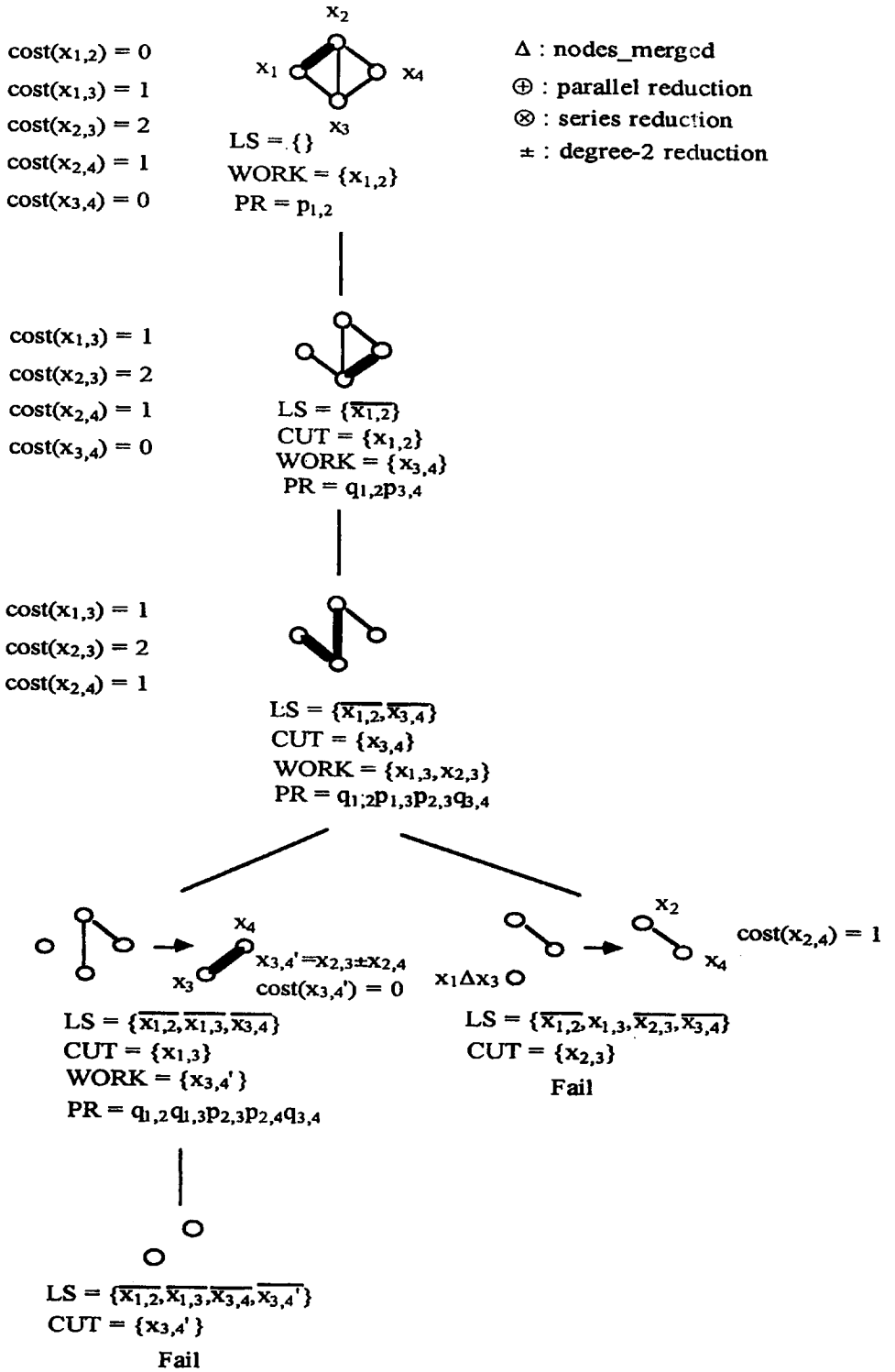

Figure 6. The splitting snapshots of the DPR1 in Figure 1.

reliability-preserving, like those incorporated in the FST-SPR algorithm. Therefore, the HRFST algorithm is also correct. ∎

The K-terminal reliability problem has been shown to be as hard as an NP problem [8]. Unlike the time complexity analysis in the K-terminal reliability problem [9,10], which is statically dependent on the given k-terminal connection, the time complexity of the distributed program

Table 1. The file distribution of the DPS.

Node	Files
x_1	F1
x_2	F2
x_3	F3
x_4	F4
x_5	F5
x_6	F6

Table 2. The program distribution of the DPS.

Node	Programs
x_1	P1
x_2	P4
x_3	P2,P3
x_4	P2,P3
x_5	P4
x_6	P1

Table 3. The data files required for execution of each program.

Program	Files Required
P1	F1,F3,F4,F5
P2	F2,F4,F5,F6
P3	F1,F3,F4,F5,F6
P4	F1,F2,F3,F4,F5,F6

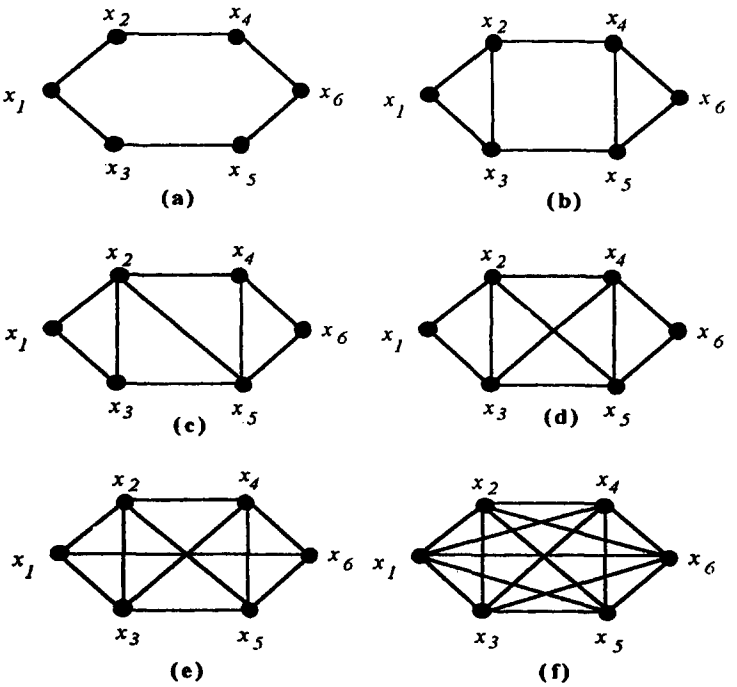

Figure 7. The six kinds of topologies for the six nodes.

reliability problem is dynamically bound to the data files required for each distributed program. The time complexity of the MFST and FARE algorithms presented in [1,2,4,5] is $O(2^m)$ in the worst case, where m denotes the number of links in the graph. However, in practical situation, such cases seldom occur, since once an MFST is found the tree expansion is stopped. The proposed HRFST algorithm uses the graph heuristic cutting technique and incorporates reduction techniques to speed up subgraph generation. The time complexity is quite difficult to quantify since the number of links and nodes may be reduced or merged during the evaluation process. However, by common reasoning, the complexity should be less than that of the MFST and

Table 4. The number of subgraph vs. different topology.
$P1$ execute at x_1.

Topology Algorithm	a	b	c	d	e	f
FST-SPR	7	13	21	30	104	314
HRFST	6	11	17	23	66	179

$P2$ execute at x_4.

Topology Algorithm	a	b	c	d	e	f
FST-SPR	7	11	11	30	104	311
HRFST	6	9	9	19	65	175

$P3$ execute at x_3.

Topology Algorithm	a	b	c	d	e	f
FST-SPR	9	29	49	74	139	402
HRFST	8	26	39	53	92	223

$P4$ execute at x_2.

Topology Algorithm	a	b	c	d	e	f
FST-SPR	11	37	56	88	180	471
HRFST	10	31	40	58	105	266

Table 5. Eight sets of data file distributions.

Set Node	Set 1 Files	Set 2 Files	Set 3 Files	Set 4 Files	Set 5 Files	Set 6 Files	Set 7 Files	Set 8 Files
x_1	none	F4	F4,F5	F1,F3	F1,F5	F3,F6	F2,F5	F3,F6
x_2	F1	F1	F2	F1,F5	F2,F4	F4	F1,F2	F1,F4
x_3	F4	F2,F3	F2,F4	F1,F6	F3	F1,F2	F4	F2,F5
x_4	F2,F5	F3,F5	F1,F3	F2,F4	F1,F3	F4,F5	none	none
x_5	F2,F3	F2,F6	F3	F2	F4,F6	none	F3,F6	F2
x_6	F5,F6	F1	F1,F6	F3	F1	F2	F5	F1,F4

FARE algorithms. One good way to compare the proposed HRFST algorithm with the FST-SPR algorithm is based on the intermediate trees (or subgraphs) generated during the entire reliability evaluation process. In this way, we can tell how much memory space and time is required for the different algorithms to run the distributed programs.

4.3. The Effects of Reliability Factors on the Performance of Different Algorithms

The file distribution, program distribution and topology of a graph all play an important role in the process of analyzing the reliability of the DPS. Those are the factors that determine the performance of reliability algorithms. In this section, these factors are used to compare two algorithms: FST-SPR and HRFST.

4.3.1. The effects of different topologies

Suppose a DPS contains six processing elements and the file distribution, program distribution, and the necessary files for each program to be executed are as listed in Tables 1–3. In Figure 7, there are six different kinds of topologies to run the DPRi ($i = 1, 2, 3, 4$) according to the distributions in Tables 1–3.

Each program is run from some specified sites to communicate with its required data files. The number of subgraphs generated in each of the six topologies for each program on different sites is shown in Table 4. The comparisons in Table 4, show clearly that the HRFST algorithm is better in different topologies.

4.3.2. The effects of different file distributions

To evaluate the influence of the file distribution on the performance of different algorithms, eight sets of file distributions were selected at random. They are listed in Table 5. A six-node topology is shown in Figure 8 for the eight sets of file distributions to reside in. The program distribution and files required for each program to be executed are presented in Tables 2 and 3 above, respectively. The comparison results are depicted in Figure 9.

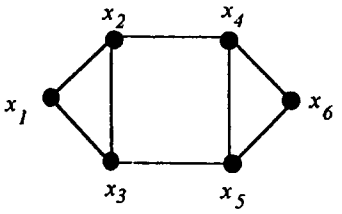

Figure 8. The topology for file distributions in Table 5.

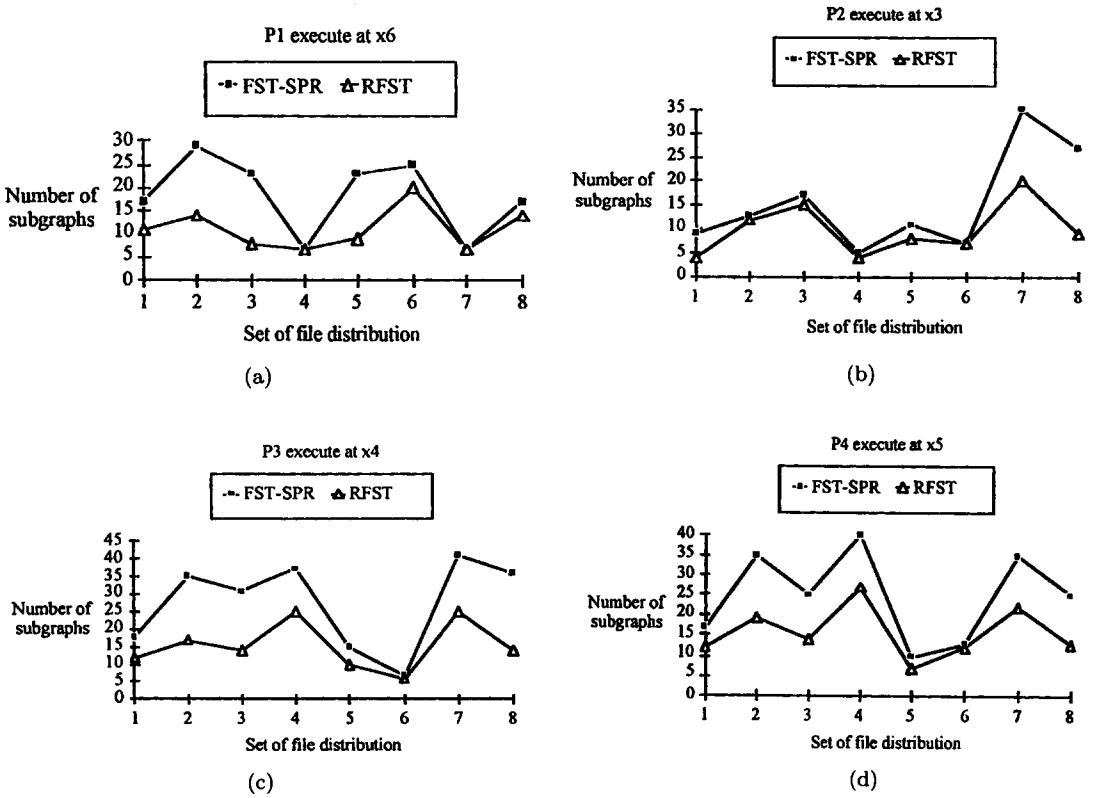

Figure 9. Number of subgraphs vs. different file distributions.

4.3.3. The effects of different program distributions

The effects of programs running on different nodes of the DPS in Figure 8 are shown in Figure 10. The file distribution and data files required for each program to be executed are presented in Tables 1 and 3 above, respectively.

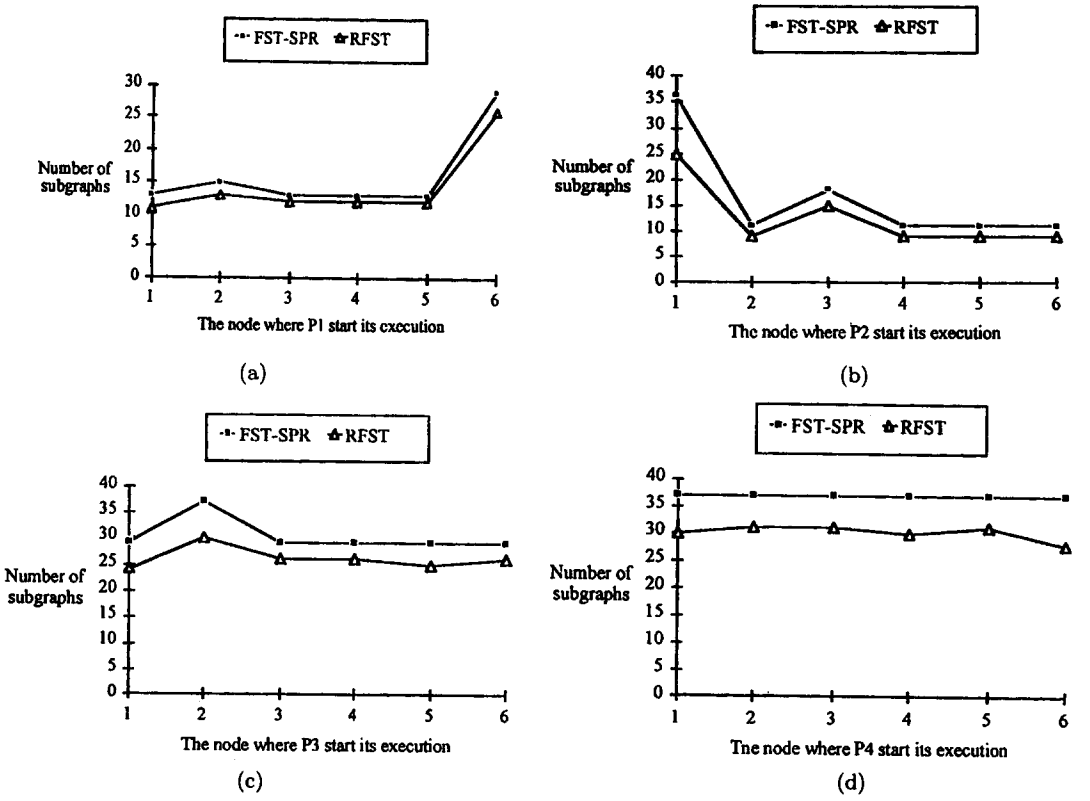

Figure 10. The number of subgraphs vs. different program distributions.

4.4. Comparison of Algorithms

4.4.1. The performance of different algorithms on complex DPS

The comparison of the performance of different algorithms on complex DPS can be viewed as an objective result, and the more complex the DPS is, the more significant the comparison result is. An eight node DPS for computing the DPR1, DPR2, DPR3, and DPR4, where the probability of each link is 0.9, is shown in Figure 11. The number of subgraphs generated by different algorithms is depicted in Table 6.

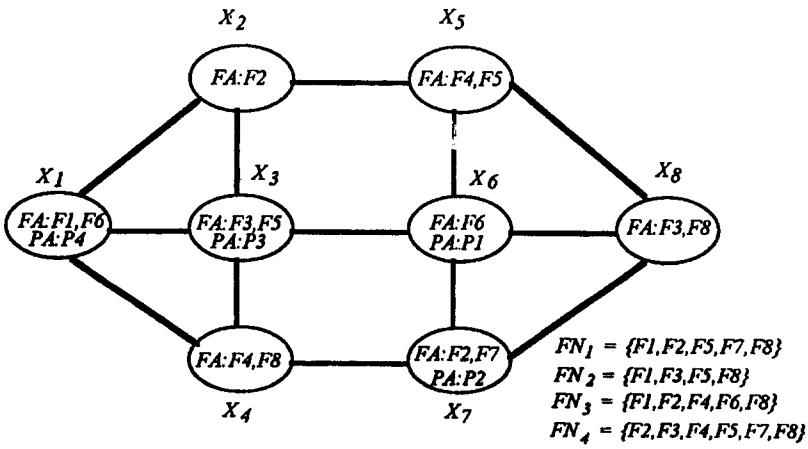

Figure 11. A complex DPS with eight processing elements.

A more complex DPS that is a well-known computer network—ARPA NET—is shown in Figure 12. There are 21 nodes and 26 links in ARPA NET. Suppose the number of data files

Table 6. The number of subgraphs vs. different programs to be executed.

Program Algorithm	P1	P2	P3	P4
FST-SPR	445	334	309	389
HRFST	113	124	118	140
DPR	0.9961182	0.9963265	0.9984532	0.9923256

is 16 and the number of programs is four. Then the file distribution, program distribution, and files required for each program to be executed are as listed in Tables 7, 8, and 9, respectively. All the subgraphs generated for computing the reliability of each program are depicted in Table 10.

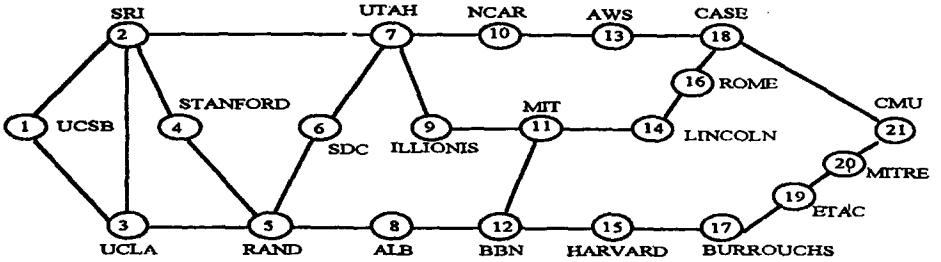

Figure 12. ARPA NET.

Table 7. The file distribution of ARPA NET.

Node	Files	Node	Files	Node	Files
x_1	F2	x_8	F12	x_{15}	F4
x_2	F3,F11	x_9	F4	x_{16}	none
x_3	F7	x_{10}	F8	x_{17}	F14
x_4	F10	x_{11}	F7	x_{18}	F15
x_5	F6	x_{12}	F16	x_{19}	F1,F9
x_6	F13	x_{13}	F10	x_{20}	F5
x_7	F12	x_{14}	F11	x_{21}	F1,F2

Table 8. The program distribution of ARPA NET.

Node	Programs	Node	Programs	Node	Programs
x_1	P3	x_8	none	x_{15}	none
x_2	P1	x_9	none	x_{16}	none
x_3	none	x_{10}	none	x_{17}	none
x_4	none	x_{11}	none	x_{18}	P2
x_5	none	x_{12}	P4	x_{19}	none
x_6	none	x_{13}	none	x_{20}	none
x_7	none	x_{14}	none	x_{21}	none

Table 9. The files for each program to be executed.

Program	Files Required
P1	F1,F3,F5,F7,F9,F11,F13,F15
P2	F2,F4,F6,F8,F10,F12,F14,F16
P3	F2,F3,F4,F5,F12,F13,F14,F15
P4	F1,F6,F7,F8,F9,F10,F11,F16

To compare the actual execution time, we present a DPRi ($i = 1, 2, 3, 4$) analysis using an IBM RISC System/6000 to collect execution time. All five algorithms are structured to have the same I/O activities to ensure the fairness of the comparison. These five programs are listed in the

Table 10. The number of subgraphs generated for each program reliability of ARPA NET.

Program Algorithm	P1	P2	P3	P4
FST-SPR	807	11598	2922	2846
HRFST	356	1274	891	691

appendix. It is clear that the HRFST algorithm outperforms the MFST algorithm. This result justifies our hypothesis that the tedious and time-consuming procedures of checking replicated trees and removing them from the TRY-LIST dominate the overall computation time. The computation times (in seconds) of the DPRi are listed in Table 11.

Table 11. The computation time (computing in seconds) of each DPRi ($i = 1, 2, 3, 4$).

Program Algorithm	P1	P2	P3	P4
FST-SPR	2.4	37.15	8.67	7.8
HRFST	1.38	6.46	3.86	3.1

4.4.2. Reliability analysis of two or more programs executed simultaneously.

To evaluate the results of reliability problem statement 2, based on the DPS in Figure 11, suppose the reliability of each link is 0.9 and several combinations of two or more programs running at the same time are chosen. The number of subgraphs generated by different algorithms is shown in Table 12.

Table 12. The number of subgraphs generated when executing two or more programs together.

Program Algorithm	P1 & P2	P1 & P3	P1 & P4	P2 & P3	P2 & P4
FST-SPR	503	446	563	329	389
HRFST	137	172	192	118	132
DPR	0.9961181	0.9960074	0.9961172	0.9962148	0.9963256
Program Algorithm	P3 & P4	P1 & P2 & P3	P1 & P2 & P4	P2 & P3 & P4	P1 & P2 & P3 & P4
FST-SPR	329	446	563	329	446
HRFST	118	160	156	133	171
DPR	0.9962148	0.9960074	0.9961172	0.9962148	0.9960074

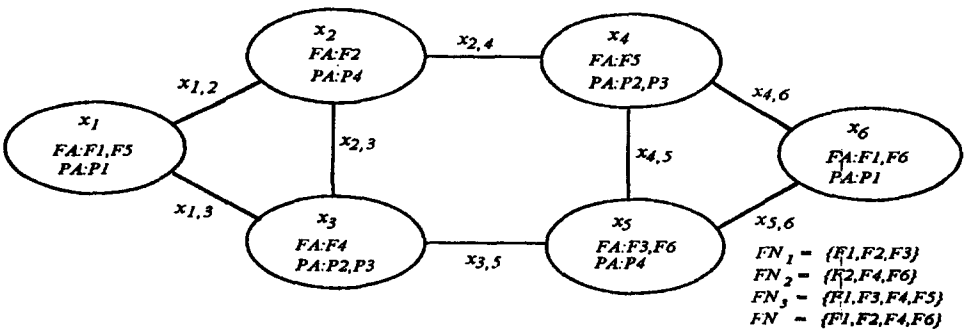

Figure 13. An example of a distributed program running from more than one site.

4.4.3. Reliability analysis of a distributed program running from more than one site

To evaluate the results of reliability problem statement 3, the Figure 13 shows an example in which there are four distributed programs and six data files in the DPS, and all four distributed programs can be executed from two different sites: P1 resides at nodes x_1 and x_6, P2 and P3 reside at nodes x_3 and x_4, and P4 resides at nodes x_2 and x_5. The results of different algorithms are compared in Table 13.

Table 13. The number of subgraphs generated for the example in Figure 13.

Program Algorithm	P1	P2	P3	P4
FST-SPR	35	11	42	46
HRFST	21	7	23	24
DPR	0.9861766	0.9854047	0.9863232	0.9853018

5. CONCLUSION

In this paper, we have presented a new reliability algorithm, called HRFST, which uses an heuristic cost evaluation function to generate an FST for analyzing the reliability of a distributed computing system. The reliability algorithm eliminates the need to search a spanning tree during the generation of each subgraph. The HRFST algorithm reduces both the number of subgraphs (or trees) generated and the actual execution time required for the reliability analysis of DPR and DSR. Our study of various sample cases and comparisons with the FST-SPR show that HRFST is more efficient than the FST-SPR algorithm.

REFERENCES

1. V.K.P. Kumar, S. Hariri and C.S. Raghavendra, Distributed program reliability analysis, *IEEE Trans. Software Eng.*, 42–50 (1986).
2. C.S. Raghavendar, V.K.P. Kumar and S. Hariri, Reliability analysis in distributed systems, *IEEE Trans. on Computer* **37** (3), 352–358 (1988).
3. S. Hariri and C.S. Raghavendra, *SYREL: A Symbolic Reliability Algorithm Based on Path and Cutset Methods*, USC Tech. Rep., (1984).
4. A. Kumar, S. Rai and D.P. Agarwal, Reliability evaluation algorithms for distributed systems, *Proc. IEEE INFOCOM* **88**, 851–860 (1988).
5. A. Kumar, S. Rai and D.P. Agarwal, On computer communication network reliability under program execution constraints, *IEEE Journal of Selected Areas in Communications*, 1393–1400 (1988).
6. D.J. Chen and T.H. Huang, Reliability analysis of distributed systems based on a fast reliability algorithm, *IEEE Trans. on Parallel and Distributed Systems* **3** (2), 139–154 (1992).
7. M.S. Lin and D.J. Chen, General graph reduction methods for the reliability analysis of distributed systems, *The Computer Journal* **36** (7), 631–644 (1993).
8. M.O. Ball, Computational complexity of network reliability analysis: an overview, *IEEE Trans. on Reliability* **R35** (3), 230–239 (1986).
9. A. Satyanarayana and K.R. Wood, A linear-time algorithm for computing K-terminal reliability in series-parallel networks, *SIAM Journal of Computing* **14** (4), 818–832 (1985).
10. K.R. Wood, Factoring algorithms for computing K-terminal network reliability, *IEEE Trans. on Reliability* **R35**, 269–278 (1986).