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 $i$
- $x_{ij}$: a link between processing elements $i$ and $j$
- $p_{ij}(q_{ij})$: probability that the link $x_{ij}$ 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)
- $F_i$: the data file $i$
- $P_i$: 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 $t$
- $FN$: 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
- $FN_t$: 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 $x_{ij} \in LS_t$
  - working if $x_{ij} \in LS_t$
  - 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 $p_{ij}$ or $q_{ij}$ 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 $PA_j$ ($j = 1, 2, \ldots, 4$).

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.

Figure 1. A simple DPS with four processing elements.
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 \( P_j \). Then the operation of links \( L_1, L_2, L_3 \) in graph A will be enough for successful execution of program \( P_j \). 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.](image)

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, \(\text{cost}(x_{i,j}) = 0\) 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.](image)

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.](image)

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 \(x_2\), 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
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 $x_{i,j} \in WORK_t$, multiply $q_{i,j}$ if $x_{i,j} \in LS_t$). 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

$t = \text{original graph} ;$

TRY = \{t\} ;
FOUND = \emptyset ;
$LS_t = \emptyset ;$
$FN = FN_i \cup FN_i (\text{where program } i \in PN) ;$

Figure 5. The probability space of the portion $B2$. 
\( R = 0 \);
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 \( x_{i,j} \in WORK_t \);
    \( PR_t = PR_t * p_{i,j} \);
    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
\( WORK_t = f \)
repeat
  \( child \leftarrow t ; \)
  \( PR_{child} = PR_t ; \)
  \( LS_{child} = LS_t ; \)
  \( child \leftarrow \text{nodes\_merged}(child, WORK_t) ; \)
  for all \( x_{i,j} \in WORK_t \) do
    \( PR_{child} = PR_{child} * p_{i,j} ; \)
  od
  \( x_{i,j} \leftarrow \text{min\_cost\_link}(child) ; \)
  \( child \leftarrow \text{cut\_link}(child, x_{i,j}) ; \)
  \( CUT_{child} = \{x_{i,j}\} ; \)
  \( PR_{child} = PR_{child} * q_{i,j} ; \)
  \( LS_{child} = LS_{child} \gg \{x_{i,j}\} \)
\( WORK_t = WORK_t \gg \{x_{i,j}\} ; \)
until (cost\((x_{i,j}) = 0) \)
return(child)
end

procedure compute_cost(t)
begin
for each link $x_{i,j}$ in subgraph $t$ do
\[
cost(x_{i,j}) = \#(FN - (FA_i \cup FA_j)) + \#(PN - (PA_i \cup PA_j))
\]
od
end

procedure modify_cost(t, $x_{i,j}$)
begin
\[
cost(x_{i,j}) = \#(FN - (FA_i \cup FA_j)) + \#(PN - (PA_i \cup PA_j))
\]
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
Reliability Analysis of Distributed Computing Systems

end

procedure nodes_merged(t, old)
begin
    for all link $x_{i,j} \in \text{old}$
        /* 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 HRFST 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, \( FN = \{F_1, F_2, F_3, F_4\} \). Applying the same algorithm, we obtain

$$\text{DSR} = P_{1,2} P_{2,3} + P_{1,2} P_{1,3} P_{2,3} + P_{1,2} Q_{1,3} P_{2,4} P_{3,4} + Q_{1,2} P_{1,3} P_{2,3} + Q_{1,2} P_{2,3} P_{2,4} P_{3,4} + Q_{1,2} Q_{1,3} P_{2,3} P_{2,4}.$$  

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

$$R(G) = p_{i,j} p_{k,l} \cdots p_{u,v} R(G_n) + p_{i,j} p_{k,l} \cdots p_{u,v} q_{u,v} R(G_n') + p_{i,j} p_{k,l} \cdots q_{u,v} R(G_{n-1}) + \cdots + q_{i,j} R(G_1').$$

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
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 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.

<table>
<thead>
<tr>
<th>Node</th>
<th>Files</th>
</tr>
</thead>
<tbody>
<tr>
<td>x1</td>
<td>F1</td>
</tr>
<tr>
<td>x2</td>
<td>F2</td>
</tr>
<tr>
<td>x3</td>
<td>F3</td>
</tr>
<tr>
<td>x4</td>
<td>F4</td>
</tr>
<tr>
<td>x5</td>
<td>F5</td>
</tr>
<tr>
<td>x6</td>
<td>F6</td>
</tr>
</tbody>
</table>

Table 2. The program distribution of the DPS.

<table>
<thead>
<tr>
<th>Node</th>
<th>Programs</th>
</tr>
</thead>
<tbody>
<tr>
<td>x1</td>
<td>P1</td>
</tr>
<tr>
<td>x2</td>
<td>P4</td>
</tr>
<tr>
<td>x3</td>
<td>P2,P3</td>
</tr>
<tr>
<td>x4</td>
<td>P2,P3</td>
</tr>
<tr>
<td>x5</td>
<td>P4</td>
</tr>
<tr>
<td>x6</td>
<td>P1</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Program</th>
<th>Files Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>F1,F3,F4,F5</td>
</tr>
<tr>
<td>P2</td>
<td>F2,F4,F5,F6</td>
</tr>
<tr>
<td>P3</td>
<td>F1,F3,F4,F5,F6</td>
</tr>
<tr>
<td>P4</td>
<td>F1,F2,F3,F4,F5,F6</td>
</tr>
</tbody>
</table>

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

The 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 situations, 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 \( x_1 \).

<table>
<thead>
<tr>
<th>Topology</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>FST-SPR</td>
<td>7</td>
<td>13</td>
<td>21</td>
<td>30</td>
<td>104</td>
<td>314</td>
</tr>
<tr>
<td>HRFST</td>
<td>6</td>
<td>11</td>
<td>17</td>
<td>23</td>
<td>66</td>
<td>179</td>
</tr>
</tbody>
</table>

P2 execute at \( x_4 \).

<table>
<thead>
<tr>
<th>Topology</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>FST-SPR</td>
<td>7</td>
<td>11</td>
<td>11</td>
<td>30</td>
<td>104</td>
<td>311</td>
</tr>
<tr>
<td>HRFST</td>
<td>6</td>
<td>9</td>
<td>9</td>
<td>19</td>
<td>65</td>
<td>175</td>
</tr>
</tbody>
</table>

P3 execute at \( x_3 \).

<table>
<thead>
<tr>
<th>Topology</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>FST-SPR</td>
<td>9</td>
<td>29</td>
<td>49</td>
<td>74</td>
<td>139</td>
<td>402</td>
</tr>
<tr>
<td>HRFST</td>
<td>8</td>
<td>26</td>
<td>39</td>
<td>53</td>
<td>92</td>
<td>223</td>
</tr>
</tbody>
</table>

P4 execute at \( x_2 \).

<table>
<thead>
<tr>
<th>Topology</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>FST-SPR</td>
<td>11</td>
<td>37</td>
<td>56</td>
<td>88</td>
<td>180</td>
<td>471</td>
</tr>
<tr>
<td>HRFST</td>
<td>10</td>
<td>31</td>
<td>40</td>
<td>58</td>
<td>105</td>
<td>266</td>
</tr>
</tbody>
</table>

Table 5. Eight sets of data file distributions.

<table>
<thead>
<tr>
<th>Set Node</th>
<th>Set 1 Files</th>
<th>Set 2 Files</th>
<th>Set 3 Files</th>
<th>Set 4 Files</th>
<th>Set 5 Files</th>
<th>Set 6 Files</th>
<th>Set 7 Files</th>
<th>Set 8 Files</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_1 )</td>
<td>none</td>
<td>F4</td>
<td>F4,F5</td>
<td>F1,F3</td>
<td>F1,F5</td>
<td>F3,F6</td>
<td>F2,F5</td>
<td>F3,F6</td>
</tr>
<tr>
<td>( x_2 )</td>
<td>F1</td>
<td>F1</td>
<td>F2</td>
<td>F1,F5</td>
<td>F2,F4</td>
<td>F4</td>
<td>F1,F2</td>
<td>F1,F4</td>
</tr>
<tr>
<td>( x_3 )</td>
<td>F4</td>
<td>F2,F3</td>
<td>F2,F4</td>
<td>F1,F6</td>
<td>F3</td>
<td>F1,F2</td>
<td>F4</td>
<td>F2,F5</td>
</tr>
<tr>
<td>( x_4 )</td>
<td>F2,F5</td>
<td>F3,F5</td>
<td>F1,F3</td>
<td>F2,F4</td>
<td>F1,F3</td>
<td>F4,F5</td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td>( x_5 )</td>
<td>F2,F3</td>
<td>F2,F6</td>
<td>F3</td>
<td>F2</td>
<td>F4,F6</td>
<td>none</td>
<td>F3,F6</td>
<td>F2</td>
</tr>
<tr>
<td>( x_6 )</td>
<td>F5,F6</td>
<td>F1</td>
<td>F1,F6</td>
<td>F3</td>
<td>F1</td>
<td>F2</td>
<td>F5</td>
<td>F1,F4</td>
</tr>
</tbody>
</table>

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.](image)

![Figure 9. Number of subgraphs vs. different file distributions.](image)

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.
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.

<table>
<thead>
<tr>
<th>Program Algorithm</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
</tr>
</thead>
<tbody>
<tr>
<td>FST-SPR</td>
<td>445</td>
<td>334</td>
<td>309</td>
<td>389</td>
</tr>
<tr>
<td>HRFST</td>
<td>113</td>
<td>124</td>
<td>118</td>
<td>140</td>
</tr>
<tr>
<td>DPR</td>
<td>0.9061182</td>
<td>0.9963265</td>
<td>0.9984532</td>
<td>0.9923256</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Node</th>
<th>Files</th>
<th>Node</th>
<th>Files</th>
<th>Node</th>
<th>Files</th>
</tr>
</thead>
<tbody>
<tr>
<td>x₁</td>
<td>F₂</td>
<td>x₈</td>
<td>F₁₂</td>
<td>x₁₅</td>
<td>F₄</td>
</tr>
<tr>
<td>x₂</td>
<td>F₃,F₁₁</td>
<td>x₉</td>
<td>F₄</td>
<td>x₁₆</td>
<td>none</td>
</tr>
<tr>
<td>x₃</td>
<td>F₇</td>
<td>x₁₀</td>
<td>F₈</td>
<td>x₁₇</td>
<td>F₁₄</td>
</tr>
<tr>
<td>x₄</td>
<td>F₁₀</td>
<td>x₁₁</td>
<td>F₇</td>
<td>x₁₈</td>
<td>F₁₅</td>
</tr>
<tr>
<td>x₅</td>
<td>F₆</td>
<td>x₁₂</td>
<td>F₁₆</td>
<td>x₁₉</td>
<td>F₁,F₉</td>
</tr>
<tr>
<td>x₆</td>
<td>F₁₃</td>
<td>x₁₃</td>
<td>F₁₀</td>
<td>x₂₀</td>
<td>F₅</td>
</tr>
<tr>
<td>x₇</td>
<td>F₁₂</td>
<td>x₁₄</td>
<td>F₁₁</td>
<td>x₂₁</td>
<td>F₁,F₂</td>
</tr>
</tbody>
</table>

Table 8. The program distribution of ARPA NET.

<table>
<thead>
<tr>
<th>Node</th>
<th>Programs</th>
<th>Node</th>
<th>Programs</th>
<th>Node</th>
<th>Programs</th>
</tr>
</thead>
<tbody>
<tr>
<td>x₁</td>
<td>P₃</td>
<td>x₃</td>
<td>none</td>
<td>x₁₅</td>
<td>none</td>
</tr>
<tr>
<td>x₂</td>
<td>P₁</td>
<td>x₉</td>
<td>none</td>
<td>x₁₆</td>
<td>none</td>
</tr>
<tr>
<td>x₃</td>
<td>none</td>
<td>x₁₀</td>
<td>none</td>
<td>x₁₇</td>
<td>none</td>
</tr>
<tr>
<td>x₄</td>
<td>none</td>
<td>x₁₁</td>
<td>none</td>
<td>x₁₈</td>
<td>P₂</td>
</tr>
<tr>
<td>x₅</td>
<td>none</td>
<td>x₁₂</td>
<td>P₄</td>
<td>x₁₉</td>
<td>none</td>
</tr>
<tr>
<td>x₆</td>
<td>none</td>
<td>x₁₃</td>
<td>none</td>
<td>x₂₀</td>
<td>none</td>
</tr>
<tr>
<td>x₇</td>
<td>none</td>
<td>x₁₄</td>
<td>none</td>
<td>x₂₁</td>
<td>none</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Program</th>
<th>Files Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>P₁</td>
<td>F₁,F₃,F₅,F₇,F₉,F₁₁,F₁₃,F₁₅</td>
</tr>
<tr>
<td>P₂</td>
<td>F₂,F₄,F₆,F₈,F₁₀,F₁₂,F₁₄,F₁₆</td>
</tr>
<tr>
<td>P₃</td>
<td>F₂,F₃,F₄,F₅,F₁₂,F₁₃,F₁₄,F₁₅</td>
</tr>
<tr>
<td>P₄</td>
<td>F₁,F₆,F₇,F₈,F₉,F₁₀,F₁₁,F₁₆</td>
</tr>
</tbody>
</table>

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
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).

<table>
<thead>
<tr>
<th>Program Algorithm</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
</tr>
</thead>
<tbody>
<tr>
<td>FST-SPR</td>
<td>2.4</td>
<td>37.15</td>
<td>8.67</td>
<td>7.8</td>
</tr>
<tr>
<td>HRFST</td>
<td>1.38</td>
<td>6.46</td>
<td>3.86</td>
<td>3.1</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Program Algorithm</th>
<th>P1 &amp; P2</th>
<th>P1 &amp; P3</th>
<th>P1 &amp; P4</th>
<th>P2 &amp; P3</th>
<th>P2 &amp; P4</th>
</tr>
</thead>
<tbody>
<tr>
<td>FST-SPR</td>
<td>503</td>
<td>446</td>
<td>563</td>
<td>329</td>
<td>389</td>
</tr>
<tr>
<td>HRFST</td>
<td>137</td>
<td>172</td>
<td>192</td>
<td>118</td>
<td>132</td>
</tr>
<tr>
<td>DPR</td>
<td>0.9961181</td>
<td>0.9960074</td>
<td>0.9961172</td>
<td>0.9962148</td>
<td>0.9963256</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Program Algorithm</th>
<th>P3 &amp; P4</th>
<th>P1 &amp; P2 &amp; P3</th>
<th>P1 &amp; P2 &amp; P4</th>
<th>P2 &amp; P3 &amp; P4</th>
<th>P1 &amp; P2 &amp; P3 &amp; P4</th>
</tr>
</thead>
<tbody>
<tr>
<td>FST-SPR</td>
<td>329</td>
<td>446</td>
<td>563</td>
<td>329</td>
<td>446</td>
</tr>
<tr>
<td>HRFST</td>
<td>118</td>
<td>160</td>
<td>156</td>
<td>133</td>
<td>171</td>
</tr>
<tr>
<td>DPR</td>
<td>0.9962148</td>
<td>0.9960074</td>
<td>0.9961172</td>
<td>0.9962148</td>
<td>0.9960074</td>
</tr>
</tbody>
</table>

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 x1 and x6, P2 and P3 reside at nodes x3 and x4, and P4 resides at nodes x2 and x5. The results of different algorithms are compared in Table 13.

<table>
<thead>
<tr>
<th>Program Algorithm</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
</tr>
</thead>
<tbody>
<tr>
<td>FST-SPR</td>
<td>35</td>
<td>11</td>
<td>42</td>
<td>46</td>
</tr>
<tr>
<td>HRFST</td>
<td>21</td>
<td>7</td>
<td>23</td>
<td>24</td>
</tr>
<tr>
<td>DPR</td>
<td>0.9861766</td>
<td>0.9854047</td>
<td>0.9863232</td>
<td>0.9853018</td>
</tr>
</tbody>
</table>

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