Terminal-Pair Reliability Using Flow Augmenting Path Search Algorithm

Moon Soo Choi and Chi-Hyuck Jun

Department of Industrial Engineering
Pohang Institute of Science and Technology
P.O. Box 125, Pohang, 790-600 Korea

ABSTRACT

This paper considers a reliability problem as a special type of flow problem and presents an algorithm to evaluate the exact 2-terminal reliability of networks by using a backtracking technique. It employs a polygon-to-chain reduction in addition to series and parallel reduction techniques to reduce execution time. In comparisons, it presents a much better performance than other algorithms known to us. We also propose a methodology to apply the algorithm for approximation of the system reliability.

I. INTRODUCTION

Over the last several decades, many reliability analysis techniques have been introduced for a variety of complex network systems. Some of them are used for approximation by Monte Carlo simulation while others are used for an exact system reliability evaluation. Fishman [5] presents the variance reduction techniques to simulate network reliability using edge-disjoint minimal cut sets and minimal path sets. Jun and Ross [6] demonstrate the power of their variance reduction techniques using the total hazard in estimating system reliability by simulation. These variance reduction techniques significantly reduce computational time in comparison with raw simulation. However, these still need a long simulation time to obtain reliable results and require additional work to derive minimal cut sets or path sets.

Although computation of the exact reliability generally requires exponential time, many authors have proposed a variety of algorithms to obtain an exact reliability. Page and Perry [8, 9] and Park and Cho [10] utilize the factoring theorem

to compute an exact system reliability by introducing recursive algorithms. Yoo and Deo [12] improve the Dotson algorithm [4] by utilizing the edge incidence matrix. Deo and Medidi [3] demonstrate the recursive algorithms for terminal-pair reliability by using a parallel processing microcomputer. Ahmad [2] presents an algorithm using backtracking techniques to evaluate the exact system reliability and Lee [7] proposes an algorithm to compute reliability of flow network considering flow capacity. The algorithms introduced above employ a factoring theorem although it is not mentioned explicitly sometimes. However, it is hard to find the best pivoting arcs of a given network which give us the least complexity.

We propose the backward factoring algorithm using backtracking techniques, (we call it flow augmenting path search algorithm (FAPSA)), to evaluate the exact system reliability. The backtracking technique provides advantages of having simple data structures to construct the search tree and of requiring a smaller memory size as compared to other algorithms. Furthermore, the algorithm shows a much better performance in comparisons with other algorithms. The algorithm is easily implemented by inserting a flow concept in the reliability problem (refer to Section 3).

II. NOTATIONS

G	network (system)
$R(G),\overline{R}(G)$	system reliability and unreliability (=1-$R(G)$) of a network G
s,t	source and sink nodes
x_i	state of the edge i (1 if it is functioning and 0, otherwise)
p_i,q_i	success and failure probabilities of the edge i
λ_i,Ω_i	remaining flow at node i, and multiplying factor which results from polygon-to-chain reduction for the remaining flow at node i
F,L	total flow which is sent to the sink, total flow loss
F_i	total flow which can be sent to sink with remaining flow at node i
L_i	total flow loss for the remaining flow at node i
f_i	flow quantity which is sent to sink through the i^{th} found path
l_j	flow loss associated with the j^{th} cut

III. FLOW AUGMENTING PATH SEARCH ALGORITHM

1. Basic Concept of the Algorithm

Let us consider the following flow problem with losses. We try to send the unit flow from the source node s to the sink node t. The edge failure probabilities are regarded as loss coefficients and so some of the flow are lost in a network. Suppose

that two nodes i and j in a network are connected by an edge k with failure probability q_k. When we send the amount of flow Q_i from node i to node j through the edge k, the actual amount of flow sent to node j is $Q_i p_k$ and the flow loss $Q_i q_k$ is assumed to remain at node i (the tail node of arc k). Under the assumptions that the remaining flow can not be sent through the same edge again and that there is no loss when sending the remaining flow backwards, (to the previous nodes on path), to augment through other paths, the remaining flow can then be sent to the sink if there is any other available path, (we call it "flow augmenting path"), but it will be permanently lost, otherwise. Thus the remaining flow of $Q_i q_k$ at node i can be sent to the sink through other paths if any exist. Finally the total flow sent to the sink will be the system reliability and the total (permanent) flow loss will be interpreted as the system unreliability.

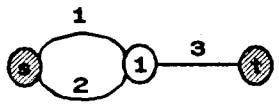


Figure 1. A Series-Parallel System

For example, consider a series-parallel system shown in Figure 1. Starting with the unit flow at node s, we can initially send p_1 to node 1 through edge 1 and then send p_1p_3 to node t through edge 3. So the remaining flow $(1\text{-}p_1)$ occurs at node s and $p_1(1\text{-}p_3)$ at node 1. The remaining flow at node 1 is lost since there is no other augmenting path to reach node t, whereas the remaining flow at node s can again be sent to node t through the augmenting path (edges 2 and 3). Therefore we send extra flow $(1\text{-}p_1)p_2p_3$ to node t. The remaining flow at node s is updated to $(1\text{-}p_1)(1\text{-}p_2)$ and to $(1\text{-}p_1)p_2(1\text{-}p_3)$ at node 1. There are no other augmenting paths, thus the total flow sent to node t (i.e. system reliability) and the total flow loss (i.e. system unreliability) are as follows:

$$R(G) = p_1p_3 + (1\text{-}p_1)p_2p_3,\ \overline{R}(G) = p_1(1\text{-}p_3) + (1\text{-}p_1)(1\text{-}p_2) + (1\text{-}p_1)p_2(1\text{-}p_3).$$

In one sense our reliability problem can be a special max-flow problem with losses under the assumptions explained previously.

2. Network Reductions

To reduce the computational burden, FAPSA uses four types of graph reduction techniques, (serial reduction, parallel reduction, a polygon-to-chain reduction, and a

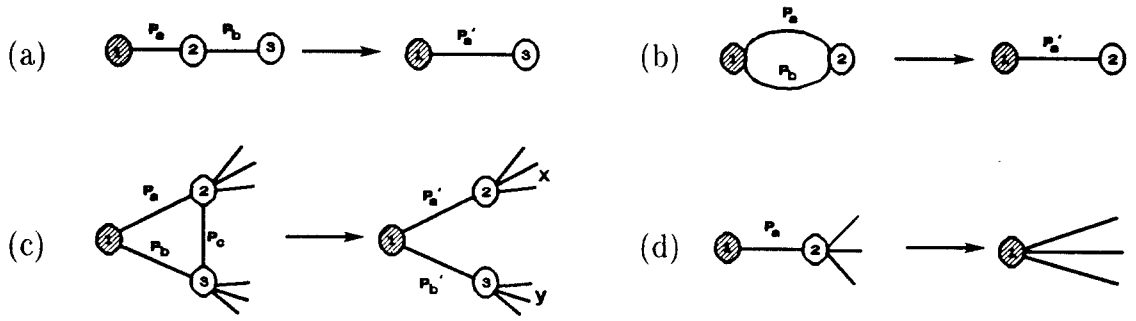


Figure 2. Reductions Employed

special type of series reduction), as shown in Figure 2. For the simple series reduction in Figure 2.(a), the new edge probability p_a' is $p_a p_b$. For the parallel reduction in Figure 2.(b) p_a' is $p_a+p_b-p_a p_b$. Figure 2.(c) presents one type of the polygon-to-chain reductions introduced in [11].

The polygon-to-chain reduction needs some explanation in a flow concept context. In Figure 2.(c), let x and y denote the flow quantities which we send to the sink node through the edges from node 2 and node 3, respectively, then the total flow in the reduced graph G' is as follows:

$$R(G') = p_a'x + (1-p_a'x)p_b'y \tag{1}$$

The new edge probabilities reduce to (see [11]) $p_a'=\delta/(\delta+A)$ and $p_b'=\delta/(\delta+B)$, respectively, where $\delta=p_a p_b+p_a p_c+p_b p_c-2p_a p_b p_c$, $A=p_b(1-p_a)(1-p_c)$, and $B=p_a(1-p_b)(1-p_c)$. The reliability of the original graph is obtained by

$$R(G) = \Omega\, R(G'), \tag{2}$$

where $\Omega=(\delta+A)(\delta+B)/\delta$. Thus for the remaining flow λ_1 at node 1, the flow quantity which we can send to the sink (which will be denoted by F_1) is

$$F_1 = \lambda_1 \Omega_1 F_1' \tag{3}$$

where F_1' denotes the flow quantity from node 1 after the reduction, and $\Omega_1=(\delta+A)(\delta+B)/\delta$. Here $\lambda_1\Omega_1$ can be considered as the new remaining flow at node 1 in the reduced graph. Then the flow loss at node 1 can be expressed by

$$L_1 = \lambda_1(1-\Omega_1 F_1') = \lambda_1(1-\Omega_1) + \lambda_1\Omega_1(1-F_1'). \tag{4}$$

Actually we do not know the flow loss L_1 until F_1' is computed. However, we are able to calculate in advance the quantity of the first term from equation (4). We consider $\lambda_1(1-\Omega_1)$ as a partial flow loss due to this reduction.

Lastly, Figure 2.(d) depicts a special type of series reduction. To remove an edge, we just update the remaining flow at node 1. Then

$$F_1 = \lambda_1 p_a F_1', \text{ and } L_1 = \lambda_1(1-p_a) + \lambda_1 p_a(1-F_1'). \tag{5}$$

The remaining flow at node 1 is updated to $\lambda_1 p_a$, and we lost the flow quantity of $\lambda_1(1\text{-}p_a)$ by this reduction. This reduction is not critical to reduce computation time but it is useful when we want to have the reliability bounds since we can update the total flow loss to have a tighter upper bound.

The authors [3, 8, 9] introduce several additional reduction rules besides the above four reductions. FAPSA excludes the self-cycling edges that all selected edges as elements of a path are functioning and it detects irrelevant edges as a part of path searching.

3. Summary of Algorithm

Initially the algorithm reduces the given graph by series and parallel reduction techniques. Then it travels to find flow augmenting paths from the source to the sink. To build the back search tree which is the LIFO (last in first out) set, find all edges connected to the source and then do reductions until no more reductions are possible. Those reduced edges are included in back search tree, and the last edge is selected from it. Assume that the selected edge is functioning, we find edges connected to the head node of the selected edge excluding self-cycling edges. They actually consist of unselected edges which are connected to the tail node, (this time it is regarded as the new source), of the selected edge and the newly found edges connected to the head node of the selected edge. After another graph reduction, the reduced edges enter search tree as the ones connected to the head node of the selected edge. This procedure is repeated until a path which connects the source and the sink is found. It is pretty efficient since this considers just the previous state and new entering edges. Thus FAPSA does reductions sequentially whenever a new edge is selected as an element of a path and it does not perform reductions for whole graph.

As explained previously, FAPSA employs the backtracking technique and the arbitrary $s-t$ path searching rule constructing the search tree simultaneously with edge reductions. On the other hand, the Reduce & Partition in [3], which is known as the fastest algorithm, uses the recursion method and the $s-t$ shortest path searching rule. The differences between the methodologies of the two algorithms fairly influence execution time.

The recursion method used in [3, 8, 9] may simplify the algorithms but it requires more memory to store each sub-problem generated by the pivoting edges. Also the edge reductions with it are inefficient since it duplicates its work when it reduces the sub-problems generated sequentially from the states of edges on a path. The shortest

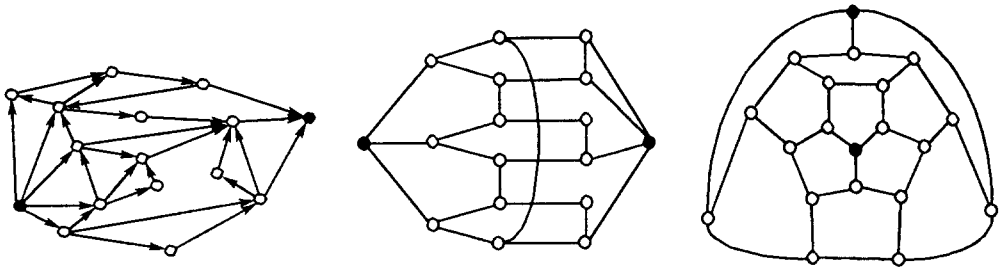
$s-t$ path searching rule in [3] generates fewer sub-problems generally without reductions, but this may not be true for reductions. When reduction techniques are employed, the best path searching rule could be the one which makes it possible to reduce the graph efficiently and thus generates fewer sub-problems eventually. Reduce & Partition searches the shortest $s-t$ path initially, then it pivots each edge on the path sequentially and recursively. Thus the pivoting edges are fixed once the shortest path is found. It may not be efficient since the rest of the edges on the path do not always constitute the shortest path of the graph generated by assuming that the previous edges on the path edge are contracted.

We avoid this inefficiency by using the backtracking method. Since FAPSA constructs the back search tree simultaneously with the edges reduced sequentially when it searches a path, it takes advantage of the previous information fully. That is, edge selection and graph reductions are done using the most recently updated graph which saves time avoiding unnecessary work. Thus FAPSA can generate a fewer number of total paths (events in [3]) eventually. In addition, the algorithm has the advantage that it detects failure events, introduced in [12], without difficulty when it travels to search a path.

IV. COMPUTATIONAL EXPERIENCE

To compare the performance of FAPSA with other algorithms, three typical benchmark networks shown in Figure 3 are used here. The computational speed of modern computers has been increased surprisingly. Thus it is meaningless to compare the performance of algorithms under different environment. Here we chose the IBM-AT personal computer, (main processor: 80286, co-processor: 80287, clock speed: 16 Mhz), for comparisons with other algorithms since it is comparable to the Macintosh II (main processor: MC68020) which is used in [3, 9]. FAPSA is programmed in C programming language and it is compiled by Turbo-C version 2.0 for the comparison, and the floating type used is *double* to deal with real numbers.

In Table 1, the computation times of three versions of FAPSA are compared with those of Modified Dotson [12], PP-F2TDN [9], Reduce & Factor [8], and Reduce & Partition [3] algorithms. The parenthesized numbers for the FAPSAs in the table represent the number of paths found, (the number of events in [3]), to evaluate the exact system reliability. These numbers can be compared with the number of events in [3]. The results of Reduce & Partition algorithm in Table 1 are the case where one processor is employed.



$R(G)$=0.997186288578 (a) $R(G)$=0.998058891278 (b) $R(G)$=0.997120396388 (c)

Figure 3. Examples for Comparisons

Table 1. Computation Times (sec) of FAPSAs and Other Algorithms

Algorithm	Computer	Language	Ex.(a)	Ex.(b)	Ex.(c)
Modified-Dotson	H. DSP-66	FORTRAN	68.9		
Reduce & Factor	Mac. plus	Pascal		61	1200
PP-F2TDN	Mac. II	Pascal	1.22	15	
Reduce & Partition	BBN Butterfly	C	0.55(82)	7.9(1120)	
FAPSA-1	IBM-AT	C	14.18(14338)	17.63(20066)	420(373091)
FAPSA-2	IBM-AT	C	0.11(74)	0.94(869)	17.24(12432)
FAPSA-3	IBM-AT	C	0.11(74)	0.88(590)	13.57(6895)

FAPSA-1 does not use any reduction techniques, FAPSA-2 applies just parallel and series reduction techniques, and FAPSA-3 uses all the types of reductions shown in Figure 2. If we compare FAPSA-1 with the Modified Dotson algorithm, and FAPSA-2 with PP-F2TDN, Reduce & Factor, and Reduce & Partition [3] algorithms which mainly apply series and parallel reduction techniques, the results show that FAPSA gives us much better performance than other algorithms. The number of paths found by FAPSA-2 is smaller than that of Reduce & Partition though both algorithms employ series and parallel reductions equivalently. This may come from the differences in efficiency of the methods. Thus we may say at least that the s-t shortest path searching rule in [3] is not devoted critically to reducing the execution time as compared to the time spent to find the shortest path. In our opinion, the backtracking method provides more advantages systematically by requiring a

smaller execution memory size, and by reducing the problem sizes efficiently. If a parallel computer is available, FAPSA can also be modified for the paralleled operation, which is the most attractive feature in [3], in processing flow at each node on a path.

V. AN EXAMINATION OF RELIABILITY BOUNDS

The FAPSA searches a minimal path from source to sink initially and then travels to find augmenting paths. Due to the searching behavior of the FAPSA, the algorithm also gives information on cuts and it sequentially accumulates the flcw quantities which are blocked by a cut as well as the flow quantities sent to the sink whenever an augmenting path or cut is found. Thus we have:

$$\sum_{i=1}^{a} f_i = R(G) = 1 - \sum_{j=1}^{c} l_j, \tag{6}$$

where a denotes the total number of augmenting paths and c is the total number of cuts found. Thus the system reliability bounds can be obtained by:

$$\sum_{i=1}^{a'} f_i \le R(G) \le 1 - \sum_{j=1}^{c'} l_j. \tag{7}$$

where $a' \le a$ and $c' \le c$.

When all edge success probabilities are 0.7 for the network shown in Figure 3.(b), Figure 4 shows the upper bound and the lower bound for the system reliability as the number of augmenting path increases.

Two methods of approximating system reliability can be considered:

Method 1: Skip augmenting for the remaining flow at a node on a path which is less than a given threshold value ϵ.

Method 2: Stop the evaluation when the difference between the upper and the lower bounds is less than a given precision ϵ'.

Method 1 not only reduces execution time but also provides in general tighter bounds than Method 2. On the other hand, Method 2 does not save execution time much since FAPSA moves to other nodes for augmenting after finishing augmenting for the remaining flow at the current node as explained previously. With Method 2, however, we can control the interval between upper and lower bounds. Thus Method 1 may be more proper for the reliability approximation. Table 2 presents the resulting bounds and computation time for the network shown in Figure 3.(c) (Example-c) with each edge success probability being 0.9 when Method 1 is used with given threshold values.

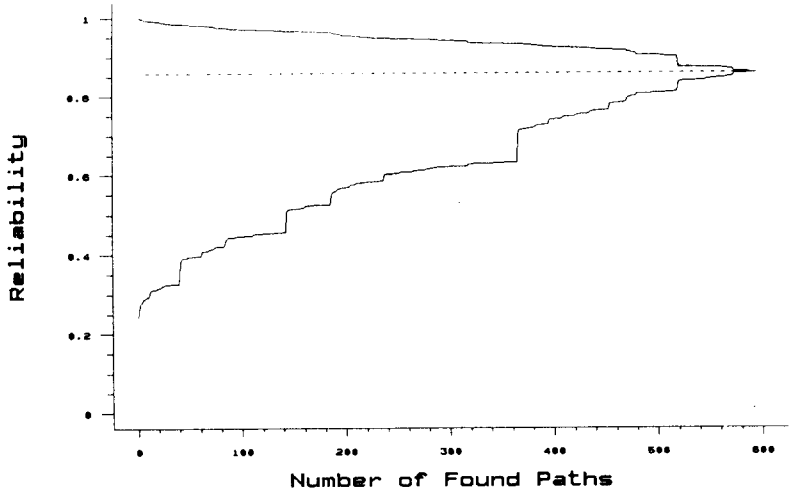


Figure 4. Lower and Upper Bounds

Table 2. Reliability Bounds for Example (c) by Method 1

Threshold ϵ	Computation Time	Reliability Bounds	
		Lower Bound	Upper Bound
10^{-10}	13.46 (6878)	0.997120395485	0.997120395485
10^{-9}	13.23 (6611)	0.997120294270	0.997120419430
10^{-8}	11.37 (5523)	0.997116811099	0.997121062645
10^{-7}	8.07 (3816)	0.997070492569	0.997127126476
10^{-6}	4.44 (1970)	0.996683754279	0.997159765971

VI. CONCLUSIONS

We presented the flow augmenting path search algorithm to evaluate the exact 2-terminal reliability of a network system. According to the types of reduction techniques employed, the algorithm is classified into three versions (FAPSA-1,2,3), for comparisons with other algorithms. The computation time of FAPSA is relatively smaller than that of other algorithms. The observed unreliability which is found sequentially may make it possible for us to control the computational time by obtaining reasonable bounds for a highly complex network.

FAPSA could be extended to the K-terminal problem by replacing a path with a spanning tree which connects K terminals. Also the algorithm can be modified for cases where reliability problems are integrated with other constraints such as capacity, distance, time, and so on.

Acknowledgement

This research was partially supported by the Korea Telecommunication Research Center (KTRC).

REFERENCES

[1] K. K. Aggarwal, "Integration of reliability and capacity in performance measure of a telecommunication network," *IEEE Trans. Reliability*, Vol R-34, 1985 June, pp 184-186.

[2] S. H. Ahmad, "A simple technique for computing network reliability," *IEEE Trans. Reliability*, Vol R-31, 1982 April, pp 41-44.

[3] N. Deo, M. Medidi, "Parallel algorithms for terminal-pair reliability," *IEEE Trans. Reliability*, Vol R-41, 1992 June, pp 201-209.

[4] W. P. Dotson, J. O. Gobien, "A new analysis technique for probabilistic graph," *IEEE Trans. Circuit and Systems*, vol 26, 1979, pp 855-865.

[5] G. S. Fishman, "A Monte Carlo sampling plan for estimating network reliability," *Operations Research*, vol 34, No. 4, 1986 Jul-Aug, pp 581-594.

[6] C. H. Jun, S. M. Ross, "Variance reduction in simulation via random hazard," *Probability in the Engineering and Informational Sciences*, 6, 1992, pp 119-126.

[7] S. H. Lee, "Reliability evaluation of a flow network," *IEEE Trans. Reliability*, Vol R-29, 1980 April, pp 24-26.

[8] L. B. Page, J. E. Perry, "A practical implementation of the factoring algorithm for network reliability," *IEEE Trans. Reliability*, Vol R-37, 1988 Aug, pp 259-267.

[9] L. B. Page, J. E. Perry, "Reliability of directed networks using factoring algorithm," *IEEE Trans. Reliability*, Vol R-38, 1989 Dec, pp 556-562.

[10] K. S. Park, B. C. Cho, "RAPID: Recursive algorithmic pivotal decomposition program for complex structural reliability analysis," *IEEE Trans. Reliability*, Vol R-37, 1988 April, pp 50-53.

[11] R. K. Wood, "Factoring algorithm for computing K-terminal network reliability," *IEEE Trans. Reliability*, Vol R-35, 1986 Aug, pp 269-278.

[12] Y. B. Yoo, Narsingh Deo, "A comparison of algorithms for terminal-pair reliability," *IEEE Trans. Reliability*, Vol R-37, 1988 June, pp 210-215.