Evaluation of Network Reliability Using Most Probable States

Dae Ho Oh[1], Dong Ho Park[2] and Seung Min Lee[2]

Abstract

An algorithm is presented for generating the most probable states in decreasing order of probability, given the reliability of each unit. The proposed new algorithm in this note is compared with the existing methods regarding memory requirement and execution time. Our method is simpler and, judging from the computing experiment, it requires less memory size than the previously known methods and takes comparable execution time to previous methods for an acceptable level of criterion.

1. Introduction

Network reliability is considered to be an important factor for assessing the effectiveness of an existing or a newly proposed network system. But exact computatiốn of network reliability is, in general, very complex and is quite often intractable mathematically. Since the size of state space increases exponentially as the size of unreliable unit increases, the evaluation of network reliability by enumerating all possible network states is useful only for a small size of network, however such evaluation seems impractical for a network with a large number of units. Therefore, it is necessary to develop an alternative way of approximating the exact network reliability or its bounds. Li and Silvester(1984) suggested "most probable" states for computing bounds of the network reliability. Since these states of high probability account for a large fraction of the total probability, this approach is considered effective in practice. Method of Li and Silvester(1984) is lacking in its flexibility since the method requires the generation of a fixed number of most probable states(MPS's) to compute the bounds. Therefore, if the number of MPS's to be generated is to be changed, then the whole process must start again from its first step. Lam and Li(1986) proposed more practical algorithm improving this disadvantage of Li and Silvester(1984)'s method. Lam and Li(1986) generated the candidate states of MPS, which is stored into data structure, called heap(Horowitz and Sahni(1976)) and then obtain the current MPS one by one in

1. Department of Computer Application, Hallym College of Information and Industry.

2. Department of Statistics, Hallym University.

decreasing order of state probability. Shier(1988) suggested more efficient algorithm than Lam and Li's method in which the MPS's are enumerated one by one from a smaller group of candidate states. Those states are generated based on the algebraic structure of network states. Gomes and Craveirinha(1998) presented an algorithm in which the MPS's are enumerated from the candidate states stored in heap using successive order. Methods of Lam and Li(1986), Shier(1988), Gomes and Craveirinha(1988) apply similar procedure that generates candidate states, and stores all the states into heap. Then, the states are rearranged in the order of state probability and then the most probable state having the greatest state probability is chosen from the heap. In this paper, an algorithm is presented for generating the most probable states in decreasing order of state probability. This method generates the candidate states in a simpler manner and prior to storing all the states into heap, a newly generated candidate state is compared with a state in the heap without computing state probability. Only those states which are acceptable are stored in the heap in the heap. By adapting this procedure, we can reduce the memory size significantly and the execution time is comparable to the existing methods for an acceptable level of criterion. In Section 2, we describe our algorithm for generating most probable states and examine its advantages and disadvantages by comparing the proposed algorithm with the other ones numerically using an example. Section 3 compares our method with Gomes et al.'s method in terms of memory requirement and empirical execution time based on several types of example networks.

Notation

Ω	state space of network
$X=(x_1,\cdots,x_n)$	a network state where, $x_i=\begin{cases}1, & \text{unit } i \text{ is operational}\\ 0, & \text{unit } i \text{ is fail}\end{cases}$
$p_i(q_i)$	probability of $x_i=1(0)$
R_i	$\frac{q_i}{p_i}$
$S=\{i_1,i_2,\cdots,i_k\}$	a set of failed unit number such that $i_1<i_2<\cdots<i_k$ where $1\le k\le n$, $1\le i_k\le n$
$S-\{i_k\}$	a set S with element i_k deleted
$S+\{i_k\}$	a set S with element i_k added
H	a heap

2. Description of Algorithm

Considering a network having n unreliable units $1, 2, \cdots, n$, we make the following assumptions:

1. each network unit can be in one of two states, operational or failed
2. the units fail independently of each other

In a binary state model, the size of Ω is 2^n. A network state is usually represented as $X=(x_1, \cdots, x_n)$, for exmple, when all the units are operating, $X=(1, 1, \cdots, 1)$. However, it may be easier to represent a network state by designating a set of failed units. Relabeling network unit as $R_1 \geq R_1 \geq \cdots \geq R_n \geq 0$, we represent each network state by the set S_i of failed units. If we let the probability $p(S_i)$ be given by

$$p(S_i) = (\prod_{i \in S} p_i)(\prod_{j \in S} q_j)$$
$$= (\prod_{i=1}^{n} p_i)(\prod_{j \in S} R_j),$$

then we have $p(S_1) \geq p(S_2) \geq \cdots \geq p(S_{2^n})$. Clearly, the most probable state is $S_1 = \phi$ and its corresponding probability is given as $p(S_1) = \prod_{i=1}^{n} p_i$. The next most probable state is $S_2 = \{1\}$ and $p(S_2) = q_1 \prod_{i=1}^{n} p_i$. On the other hand, the least probable state is $S_{2^n} = \{1, 2, \cdots, n\}$ and $p(S_{2^n}) = \prod_{i=1}^{n} q_i$. Instead of enumerating all the network states to evaluate the network reliability or performance, we consider only m most probable states $S_1, S_2, \cdots S_m$, which satisfy a certain criteria. For example, we choose the minimum m satisfying $\sum_{i=1}^{m} p(S_i) \geq c$(coverage) or some other stopping rule. When m most probable states, $S_1, S_2, \cdots S_m$, are identified, the upper and lower bounds of a given network reliability measure may be derived by applying the method of Li and Silvester. All the algorithms proposed by Lam and Li, Shier and Gomes et al. to generate the most probable states store certain candidate states in a data structure, which is referred to as a heap(Horowitz and Sahni(1976)). A heap has three major operations: Firstly, inserting the elements along with its weights, secondly, reordering elements in the heap by the weight so that the root of heap has always the largest weight and finally, selecting an element having largest weight from the root of heap. At each iteration of these algorithms, certain candidate states may be inserted into the heap and after reordering these states by the size of its probability, a state which is most probable state at the current iteration, is selected from the heap. We also adopt a heap for storing the candidate states.

Algorithm

```
For a given network unit $\{1,\cdots,n\}$,
relabeling units as $R_1 \geq R_1 \geq \cdots \geq R_n \geq 0$.
1. Initialize: $S_1 \leftarrow \phi$, $S_2 \leftarrow \{1\}$, $S_3 \leftarrow \{2\}$, $H=\phi$, $i \leftarrow 3$
2. Repeat
    2.1. generate candidate states from $S_i$
        2.1.1. if first element of $S_i > 1$ then generate $S_i + \{1\}$
        2.1.2. for each element of $S_i$, except first, generate $S_i - \{i_l\} + \{i_l + 1\}$
    2.2. checking comparability for candidate states
    2.3. insert accepted candidate states into H
    2.4. select current MPS from root of H
        $i \leftarrow i+1$, $S_i \leftarrow$ selected state
  The process continues until sufficient number of states are selected.
```

Remark **2.1** In our algorithm, step 2.2 is the procedure that checks comparability of states by taking the difference of candidate states generated in step 2.1 from each element of states in heap. Identifying the sign, we can decide to store candidate states or not. If all the sign is positive, the candidate state is considered to be comparable state even without computing its probability. When $S_i = \{2,4\}$, the generated candidate states are $\{3,4\}, \{1,2,4\}, \{2,5\}$ and if there is a state $\{1,2,3\}$ in heap, then for $\{1,2,4\}$ and $\{1,2,3\}$, the differences are, 4-3, 2-2, 1-1, since all the signs are positive, the state $\{1,2,4\}$ is not accepted and thus is not inserted into H.

Table 2.1. Comparison of heaps ($n=9$).

Iter.	output state	heap after insertions			
		Lam & Li	Shier	Gomes et al.	Ours
0	ϕ	-	-	-	-
	{1}	{2},{1,2}	{2}	{2},{1,2}	-
1	{2}	{3},{1,2},{2,3}	{3},{1,2}	{3},{1,2}	{3},{1,2}
2	{3}	{4},{1,2},{2,3},{3,4}	{4},{1,2},{1,3}	{4},{1,2}	{4},{1,2}
3	{4}	{1,2},{2,3},{5} {3,4},{4,5}	{1,2},{1,3},{1,4} {5}	{5},{1,2}	{5},{1,2}
4	{1,2}	{1,3},{2,3},{5},{3,4} {4,5},{1,2,3}	{1,3},{1,4},{5}	{1,3}{5},{1,2,3}	{1,3}{5}
5	{1,3}	{2,3},{1,4},{5},{3,4} {1,2,3},{1,3,4},{4,5}	{2,3},{1,4},{5}	{2,3},{1,4},{5}, {1,2,3}	{2,3},{1,4},{5}

Consider a 9 unit network with $p_1=0.8$, $p_2=0.85$, $p_3=0.99$, $p_4=0.9$, $p_5=0.999$, $p_6=0.999$, $p_7=0.95$, $p_8=0.999$, $p_9=0.999$ to illustrate the differences in terms of heap size. Table 2.1 presents the results up to only 5 iterations, although there are more numerical calculations done for the network.

From Table 2.1., Lam and Li generate almost two candidate states at each iteration, and thus the heap size increases very rapidly as the iterations continue. At iteration 2, Shier's algorithm generate candidate state, $\{1,3\},\{4\}$ from state $\{3\}$ and all the states remained in heap are $\{4\},\{1,2\},\{1,3\}$, and $\{1,2\}$ and $\{1,3\}$ are comparable directly without computing probability. Also, in Gomes et al.'s algorithm generate $\{2,3\}$ from state, $\{1,3\}$ at iteration 5. But, this state is directly comparable to $\{1,2,3\}$. These comparable candidate states have relatively low probabilities, and are stored into the heap at the early iteration. As seen Table 2.1, our algorithm generates only those candidate states which do not directly comparable without computing their state probability from step 2.2. From this aspect of the algorithm, our method can reduce the size of heap throughout the evaluation process.

3. Numerical Examples

In this section, we compare our method with Gomes et al.'s method with regard to heap size and empirical execution time. The following results are obtained by implementing C programming language on a microcomputer

Table 3.1. Comparing the algorithm of example 3.1-1

m	Gomes et al.	Ours	coverage (%)
50	12	5	99.01
1000	126	27	99.999
3000	214	49	99.99999
5000	234	68	99.999999

Example 3.1 case 1) Consider a network with 15 units(not shown its structure here), $p_1=0.5$, $p_2=0.6$, $p_3=0.7$ and $p_i=0.99+0.001\cdot(i-3)$, $i=4,5,\cdots,12$, $p_{13}=0.9992$, $p_{14}=0.9993$, $p_{15}=0.9994$. Table 3.1 displays the memory size of heap for each method when the value of m changes from 50 to 5000. This case shows that only three units have low reliabilities and remaining 12 units are highly reliable. The size of state space is 32768, but only a small number of states, say 50, (0.15%) yield 99.01% coverage. We note that the heap size for Gomes et al.'s method is about 4 times as large as that needed by our method.

Table 3.2. Comparing the algorithm of example 3.1-2

m	Gomes et al.	Ours	coverage (%)
1000	336	20	74.93
3000	1017	39	92.71
5000	1499	46	96.94

case 2) Consider a network with 15 units, $p_i = 0.555 + 0.001 \cdot (i-1)$, $i = 1, \cdots, 7$, $p_i = 0.7 + 0.05 \cdot (i-8)$, $i = 8, \cdots, 13$, $p_{14} = 0.97$, $p_{15} = 0.99$. This case shows that about 20% of all the units have high reliabilities. Table 3.2. shows that the coverage is quit low with $m = 1000$ MPS (3%) and the heap size for Gomes et al.'s method increases rapidly and it requires about 4 times as big heap size as ours.

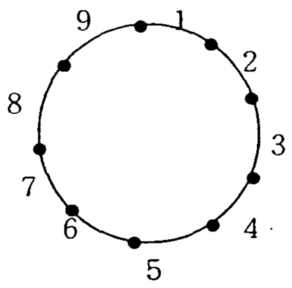

Figure 3.1. Example network

Table 3.3. units, probabilities, weight, ranks

unit i	p_i	q_i	$R_i = q_i/p_i$
1	0.8	0.2	0.25
2	0.85	0.15	0.17647
3	0.99	0.01	0.0101
4	0.9	0.1	0.11111
5	0.999	0.001	0.001
6	0.999	0.001	0.001
7	0.95	0.05	0.05263
8	0.999	0.001	0.001
9	0.999	0.001	0.001

Table 3.4. Comparing the algorithm for the network in Figure 3.1($n = 9$)

m	Gomes et al.		Ours		coverage
	heap size	time(sec.)	heap size	time(sec.)	(%)
50	9	0.0019164	4	0.0019292	99.967
100	9	0.0020631	5	0.0021013	99.998
150	10	0.0022099	5	0.0022763	99.9998
200	10	0.0023627	5	0.0024644	99.99997
300	11	0.0026994	6	0.0028750	99.999999
Total	11	0.0034678	6	0.0039167	100

Example 3.2 Consider an example network given in Figure 3.1, with its reliabilities be given in Table 3.3. Table 3.4 represents the heap size and execution time(in seconds) for two algorithms. Execution time is obtained by repeating the simulation 50 times. As seen from Table 3.4, our algorithm reduced heap size more than 40% and empirical execution time is comparable to Gomes et al's algorithm.

4. Concluding Remarks

In this paper, we suggested a new algorithm for generating the most probable states for a given network. From several numerical examples, we showed that our algorithm significantly reduces the memory size by generating the non-comparable states only. Regarding the empirical execution time, we also showed that our algorithm is comparable to the method of Gomes et al.(1988) for a generally acceptable level of coverage. Our algorithm tends to take longer execution time as m becomes large, which is due to the checking process for the candidate states being directly comparable, although the reduced heap size helps to shorten the heap operation time. Therefore, there is a room to improve our algorithm by modifying the method of checking candidate states for a more attractive achievement of execution time when m becomes large.

REFERENCES

[1] V.O.K. Li and J.A. Silvester, "Performance analysis networks with unreliable components", IEEE Transactions on Communications, vol. COM-32, pp 1105-1110, 1984.

[2] Y.F. Lam and V.O.K. Li, "An improved algorithm for performance analysis of networks with unreliable components", IEEE Transactions on Communications, vol. COM-34, pp 496-497, 1986.

[3] D.R. Shier, "A new algorithm for performance Analysis of communication systems", IEEE Transactions on Communications, vol. COM-36, pp 516-519, 1988.

[4] E.J. Valvo, D.R.Shier and R.E. Jamison, "Generating the most probable states of a communication system", Proc. IEEE INFOCOM '87, San Francisco, pp 1128-1136, 1987.

[5] T.M.S. Gomes and J.M.F. Craveirinha, "Algorithm for sequential generation of states in failure-prone communication network", IEE Proc Commun. vol. 145, pp73-79, 1998

[6] E. Horowitz and S. Sahni, "Fundamentals of data structure", Pitman, 1976.