ELSEVIER

Reliability Engineering and System Safety 77 (2002) 111–119

RELIABILITY ENGINEERING & SYSTEM SAFETY

www.elsevier.com/locate/ress

An efficient algorithm for sequential generation of failure states in a network with multi-mode components

Teresa Gomes[a,b,*], José Craveirinha[a,b], Lúcia Martins[a,b]

[a]Departamento de Engenharia Electrotécnica e de Computadores, FCTUC, Pólo II da Universidade de Coimbra, Pinhal de Marrocos, 3030-290 Coimbra, Portugal

[b]INESC–Coimbra, Rua Antero de Quental 199, 3000-033 Coimbra, Portugal

Received 11 September 2000; accepted 21 December 2001

Abstract

In this work a new algorithm for the sequential generation of failure states in a network with multi-mode components is proposed. The algorithm presented in the paper transforms the state enumeration problem into a *K*-shortest paths problem.

Taking advantage of the inherent efficiency of an algorithm for shortest paths enumeration and also of the characteristics of the reliability problem in which it will be used, an algorithm with lower complexity than the best algorithm in the literature for solving this problem, was obtained.

Computational results will be presented for comparing the efficiency of both algorithms in terms of CPU time and for problems of different size. © 2002 Elsevier Science Ltd. All rights reserved.

Keywords: Network reliability; Multi-mode components network; Failure states generation

1. Introduction

The models of reliability analysis of communications/computer networks are based on the idea of associating, in a probabilistic manner, the states of a system, with performance measures. This methodology was mainly developed by Meyer (see e.g. Ref. [1]) leading to the concept of 'performability'. An exhaustive reliability study of a network with significant number of components quickly becomes computationally unfeasible having in mind the exponential increase in the state space dimension. This problem becomes more critical as the computational cost of the performance calculation for each state increases. Li and Silvester [2] suggested that only the most probable network states, enabling a certain minimum coverage of the space state, needed to be considered and developed an algorithm for selecting those states. The efficiency of this algorithm was improved by Lam and Li [3] and later by Yang and Kubat [4], taking as a basis an algorithm [5] developed by the same authors for networks with multi-mode components. An algorithm for components with two states, with lower complexity than those algorithms was proposed by Gomes and Craveirinha [6]. For networks with multi-mode components (the focus of this paper) an algorithm generalizing the Lam and Li [3] approach was introduced by Chiou and Li [7]. The performance degradation of each component is here characterized by up to N different modes (or states) and the algorithm enumerates a pre-determined number m of network states in order of decreasing probability. Another more efficient algorithm was proposed by Yang and Kubat [5] which was shown to have an upper limit complexity $O(nN|\Omega_\varepsilon|)$ when used for generating the most probable states corresponding to the minimal set Ω_ε which satisfies the requirement of a coverage probability of the state space not less than $1-\varepsilon$, in a network with n components. It is at least n times faster than the one by Chiou and Li [7]. Another feature of this algorithm is that it does not need to 'guess' the dimension of Ω_ε: the algorithm keeps generating states until the required coverage of the state space is obtained.

In this paper, a new algorithm is proposed for efficiently generating the sequence of most probable states in a network with multi-mode components. This algorithm is based on the transformation of the enumeration problem into a *K*-shortest paths problem. The major advantage of the proposed algorithm results from its lower complexity

* Corresponding author. Tel.: +351-239-796261; fax: +351-239-796247.

E-mail addresses: teresa@dee.uc.pt (T. Gomes), jcrav@dee.uc.pt (J. Craveirinha), lucia@dee.uc.pt (L. Martins).

0951-8320/02/$ - see front matter © 2002 Elsevier Science Ltd. All rights reserved.
PII: S0951-8320(02)00006-6

and therefore lower running CPU time and also from the quite significant reduction in memory requirement as compared with the Yang and Kubat [5] algorithm.

The paper is organized as follows. Section 2 introduces basic concepts of the underlying model and previous results necessary for justifying the algorithm. The formalization of the algorithm, its complexity analysis and memory requirements are presented in Section 3. Computational results comparing, in terms of CPU time the Yang and Kubat [5] approach and the proposed algorithm can be found in Section 4 which is followed by some conclusions.

2. Foundations of the model

2.1. Basic concepts

Let us consider a network with n components where each component i can be in $j = 0, 1, 2, \dots, d(i)$ different modes, corresponding to different operational, partially operational or inoperational conditions. It is assumed that component failures are statistically independent and that the probability of any component being in state j is $p(i,j)$, such that $\sum_{j=0}^{d(i)} p(i,j) = 1$. Let $\mathrm{op}(i) = p(i,j_0)$ be the probability of the component being fully operational, then the probability of being in any inoperability state is $\mathrm{ip}(i) = 1 - \mathrm{op}(i)$. In practical situations it is only of interest to consider $1/2 \leq \mathrm{op}(i) \leq 1$, $p(i,j_0) \geq 1/2$. Nevertheless in order to maintain the algorithm as general as possible the following definitions are introduced, inspired by the definitions in Ref. [2]. A component is said to be *connected* if it is in its most probable state; otherwise it is said to be *disconnected*. Let $p(i,j_m) = \max_j p(i,j)$ and $p(i) = p(i,j_m)$ be the probability of component i being connected. If $\mathrm{op}(i) \geq \mathrm{ip}(i)$, then $p(i) = \mathrm{op}(i)$. If $\mathrm{op}(i) < \mathrm{ip}(i)$, then $p(i) = p(i,j_m)$ (probability of the most probable inoperational state). The probability of being disconnected is $\sum_{j \neq j_m}^{d(i)} p(i,j)$ and $d(i)$ represents the number of modes of disconnection. Let the probability of the disconnected states be $q(i,j)$:

$$q(i,j) = p(i,j'), \qquad \text{with } i = 1, 2, \dots, n$$

$$j' = 0, 1, \dots, j_m - 1, j_m + 1, \dots, d(i) \tag{1}$$

$$j = \begin{cases} j' + 1 & \text{if } j' < j_m \\ j' & \text{if } j' > j_m \end{cases}$$

Let S_k designate the states of the network. Then similarly to Ref. [2]:

$$P(S_k) = \prod_{i=1}^{n} p(i)^{1 - T_i(S_k)} q(i,j)^{T_i(S_k)} \tag{2}$$

where

$$T_i(S_k) = \begin{cases} 0 & \text{if } i \text{ is connected in state } S_k \\ 1 & \text{if } i \text{ is in the } j\text{th disconnected mode in state } S_k \end{cases} \tag{3}$$

It is assumed that S_1 is the state where all the network components are connected and has probability: $P(S_1) = \prod_{i=1}^{n} p(i)$.

2.2. Target graph of the algorithm

In order to specify a graph where the algorithm finds K-shortest paths in order to solve the state enumeration problem the following definitions are introduced. Let R be a vector such that $R(r)$ is associated with a given disconnected mode of some component of the network:

$$R(r) = \frac{q(i,j)}{p(i)}, \qquad i = 1, 2, \dots, n; \; j = 1, 2, \dots, d(i);$$

$$d_T(i) = d_T(i-1) + d(i); \qquad r = d_T(i-1) + 1, \dots, d_T(i) \tag{4}$$

where by assumption $d(0) = 0$, and $d_T(0) = 0$.

A function id is defined that transforms the index $v = \{1, 2, \dots, d_T(n)\}$ of R into the pair (i,j) which identifies the jth disconnected state of component i, according to Eqs. (1) and (4). Another function $\mathrm{id_e}$ is defined that transforms the index v of R into the label i of the component such that (i,j) is the pair associated with v, through Eqs. (1) and (4).

Let $S_k = \{e_1, e_2, \dots, e_w\}$ with $e_r \in \{1, 2, \dots, d_T(n)\}$ and $r = 1, 2 \dots, w$, represent a state of the network. Then:

$$\forall e_v, e_u \in S_k : \; \mathrm{id_e}(e_v) \neq \mathrm{id_e}(e_u) \tag{5}$$

and from Eqs. (2) and (4):

$$P(S_k) = P(S_1) \prod_{i=1}^{w} R(e_i) \tag{6}$$

Note that in this manner S_k is completely defined by the set of the disconnected modes (of the components) which characterize the state.

In order to obtain an additive metric required by the used shortest path algorithm the probability of each state is transformed by:

$$-\ln P(S_k) = -\ln P(S_1) - \sum_{i=1}^{w} \ln R(e_i) \tag{7}$$

where the minus sign guarantees that a positive valued metric is obtained.

A graph is then considered where paths originate at a fictitious node $s = 0$ and terminate at a fictitious node $t = d_T(n) + 1$. The intermediate nodes of a path will represent the elements of S_k if the matrix of the costs associated with the arcs is constructed in a convenient form, as analysed hereafter.

Let $V = \{s, v_1, v_2, \dots, v_{t-1}, t\}$ be the node set of a directed graph and L the arc set, composed of ordered pairs of elements in V. The kth path generated by the algorithm in this graph will be specified by the sequence $p_k = \langle s, (s, v_1), v_1, \dots, (v_w, t), t \rangle$. The corresponding state is $S_k = \{v_1, v_2, \dots, v_w\}$ where the auxiliary nodes s and t were excluded. The cost of such path will be given by:

$$c(p_k) = \sum_{u=0}^{w} c_{v_u v_{u+1}} \tag{8}$$

where $c_{v_i v_j}$ represents the cost of arc (v_i, v_j).

In order to guarantee that there is a strict mapping of state probabilities and path costs, from Eq. (7):

$$-\ln P(S_k) = -\ln P(S_1) + c(p_k) \tag{9}$$

This implies that

$$c(p_k) = -\sum_{u=1}^{w} \ln R(v_u) + c_{v_w t} \tag{10}$$

and

$$c_{v_u v_{u+1}} = -\ln R(v_{u+1}), u = 0, 1, \ldots, d_T(n) - 1 \wedge s = v_0 = 0 \tag{11}$$

$$c_{v_w t} = 0, \qquad t = v_{w+1} = d_T(n) + 1 \tag{12}$$

Next a *cost matrix* $[c_{ak}]$ of dimension $(t) \times (t+1)$ is defined such that the cost of an arc is the additional cost of introducing a new disconnected component (in a certain disconnected mode), while satisfying relations (4), (11) and (12):

$$c_{sk} = c_{0k} = -\ln R(k), \qquad \text{with } k < t \tag{13}$$

$$c_{ak} = \infty, \qquad \text{if} a \geq k \tag{14}$$

$$c_{ak} = \infty, \qquad \text{if } 0 < a < k \wedge \mathrm{id_e}(a) = \mathrm{id_e}(k) \tag{15}$$

$$c_{ak} = -\ln R(k), \qquad \text{if } 0 < a < k \wedge \mathrm{id_e}(a) \neq \mathrm{id_e}(k) \tag{16}$$

$$c_{at} = 0, \qquad \text{with } t = d_T(n) + 1 \tag{17}$$

The elements of row 0 represent the cost of passing from the most probable state S_1 to a state $S_v = \{k\}$. The element c_{st} takes the value 0 so that the shortest path $p_1 = \langle s, (s,t), t \rangle$ corresponds to state $S_1 = \emptyset$. The costs c_{at} just enable that all nodes may reach the auxiliary node t without additional cost. Relation (15) prevents the generation of paths associated with any given state S_u, containing two or more disconnection modes of the same component, since such states cannot exist. Eq. (14) prevents the generation of paths including repeated nodes and the generation of different paths formed by the same set of nodes, placed in different order. Finally, Eq. (17) implies that adding node t to a path does not have any cost, according to Eq. (12). It should be pointed out that the obtained matrix is acyclic, that is no path can be constructed with identical original and terminal nodes. Therefore, K-shortest paths algorithms will always obtain loopless paths, when applied to a graph the arcs' cost of which are given by Eqs. (13)–(17).

In order to justify that the sequential enumeration of the most probable states is equivalent to obtaining the k-shortest paths from s to t in the directed graph defined earlier, two auxiliary propositions are now presented.

Proposition 2.1. *Let p be a path from s to t:*

$$p = \langle s, (s, v_1), v_1, (v_1, v_2), v_2, \ldots, (v_w, t), t \rangle$$

obtained from a shortest path algorithm in the graph the arcs of which have the costs defined by Eqs. (13)–(17) and the arcs with ∞ *cost deleted; then* $s < v_1 < v_2 < \cdots < v_w < t$.

Proof. $\forall i,j \in [0, 1, \ldots d_T(n) + 1] : i \geq j \Rightarrow c_{ij} = \infty$, according to Eq. (14).

Proposition 2.2. *Let p be a path from s to t:*

$$p = \langle s, (s, v_1), v_1, \ldots, v_{i-1,}(v_{i-1}, v_i), v_i, \ldots, v_{j-1,}(v_{j-1}, v_j), \ldots, (v_w, t), t \rangle$$

obtained from a shortest path algorithm in the graph the arcs of which have the costs defined by Eqs. (13)–(17) and the arcs with ∞ *cost deleted; then* $\mathrm{id_e}(v_a) \neq \mathrm{id_e}(v_b)$, $\forall v_a, v_b \in p$.

Here it is conventioned that $\mathrm{id_e}(s) = 0$ *and* $\mathrm{id_e}(t) = n + 1$.

Proof. Let $v_i, v_j \in p$, then by Proposition 2.1, $v_i < v_j$.

If $(v_i, v_j) \in p$ then from Eq. (15), $\mathrm{id_e}(v_i) \neq \mathrm{id_e}(v_j)$, since otherwise the arc would have ∞ cost.

If $v_i, v_j \in p$ and $(v_i, v_j) \notin p$ then, it is possible to construct a sub-path p^* of p, from v_i to v_j :

$$p^* = \langle v_i, (v_i, v_i'), v_i', \ldots, v_j', (v_j', v_j), v_j \rangle$$

where by Proposition 2.1 $v_i < v_i' < \ldots < v_j' < v_j$. Also, by construction of R in Eq. (4), if $\mathrm{id_e}(v_i) = \mathrm{id_e}(v_j) = u$ then $\mathrm{id_e}(r) = u$, $r = v_i, v_i + 1, \ldots, v_j$

But, according to Eq. (15), the arcs in p^* may only exist if their extreme nodes (which are in the interval $[v_i, v_j]$) have different values of $\mathrm{id_e}$, therefore if $v_i, v_j \in p$ and $(v_i, v_j) \notin p$ then $\mathrm{id_e}(v_i) \neq \mathrm{id_e}(v_j)$, which concludes the proof.

An illustrative example of the calculation of R, corresponding matrix and target graph is presented in Appendix C.

3. The algorithm

Next we formalize the algorithm for enumerating the K most probable states, taking as a basis the graph defined in the previous section, the nodes of which represent the different disconnected modes of all the components; the arcs and their associated costs are defined by matrix $[c_{ak}]$ and the cost of a path, representing state S_u, when added to $-\ln P(S_1)$ gives the value $-\ln P(S_u)$. For calculating the K-shortest paths we will use algorithm MPS in Ref. [8] which is to the best of our knowledge, the most efficient algorithm available in the literature.

The variables P_c and P_d represent the calculated and the desired probability coverage, respectively.

Algorithm 3.1 ([State generation in a multi-mode component network]).

```
1. Input: $p(i,j)$, $i = 1,2,\ldots,n$, $j = 0,1,2,\ldots,d(i)$; $P_d$, and
   assume without loss of generality:
   $p(i,0) = \max_{j=0,\ldots,d(i)} p(i,j)$.
2. Calculate $P(S_1)$, $P(S_1) = \prod_{i=1}^{n} p(i,0)$.
3. Construct a vector $R$ according to Eq. (4).
4. Define the graph with arcs having associated costs $[c_{ak}]$
   given by Eqs. (13)–(17).
5. Construct the shortest tree of all nodes to $t$, $\tau_t^*$. This is
   trivial because such a tree is formed by the paths
   $\langle v,(v,t),t\rangle$ of cost 0, with $v = 0,1,2,\ldots,t-1$.
6. Obtain a representation of the graph in the sorted forward
   star form (see details in Appendix D or Dial et al. [9]).
7. $u = 0$; $P_c = 0$;
8. While ($P_c < P_d$)
     (a) $u \leftarrow u + 1$;
     (b) Calculate the next shortest path, $p_u$, from $s$ to $t$
         using the MPS algorithm; let its cost be $c(p_u)$.
     (c) Calculate the associated state probability: $P(S_u) =$
         $e^{-c(p_u)} P(S_1)$.
     (d) $P_c \leftarrow P_c + P(S_u)$
   EndWhile
```

3.1. Algorithm complexity

Firstly note that the re-labelling operations have a cost proportional to the number of relabelled elements. The complete expression of the complexity of the K-shortest path algorithm MPS is given in [8]:

$$O(|L| + |V|\log_2|V| + |L| + |L|\log_2|V| + K|V|) \tag{18}$$

where each of the terms in the above-mentioned expression can be interpreted as follows:

- $|L| + |V|\log_2|V|$: obtaining the tree of shortest paths to t, τ_t^*;
- $|L|$: calculation of reduced costs in the arcs [8,10];
- $|L|\log|V|$: storing the arcs in the sorted forward star form;
- $K|V|$: cost of obtaining the K-shortest paths, after performing the previous operations. This results from the fact that, in MPS, each time a shortest path is selected at most $|V|$ new paths have to be generated.

This gives rise to a simplified expression for the complexity of the form [8]:

$$O(|L|\log_2|V| + K|V|) \tag{19}$$

We now present the complexity calculation for our simplified version of MPS:

- Having in mind the particular structure of the cost matrix, the construction of the tree of the shortest paths from every node to t is trivial (all its arcs have null cost) and its cost in terms of complexity is $O(|V|)$.
- As the cost of any path to t is zero (in τ_t^*), it is not necessary to calculate reduced costs in the arcs (reported to τ_t^*) since they are equal to the arc costs.
- As for the construction of the graph in the sorted forward star form, it may be done in the following manner, with a complexity proportional to the number of arcs of the graph, having in mind the particular structure of the cost matrix:
  - order all the arcs originated at node $s = 0$ by decreasing cost using the quicksort algorithm which has complexity $O(|V|\log_2|V|)$ [11];
  - then define straightforwardly the remaining forward star form structure (no further sorting algorithm is needed) since the successors of the arcs originated at nodes $r > 0$ appear in the same order as for node 0, whenever they exist.

  In this manner it is possible to create the forward star form structure with complexity:

$$O(|L| + |V|\log_2|V|) = O(|L|) \tag{20}$$

  assuming $|L| \approx |V|^2$.
- Due to the fact that all nodes in V are adjacent to t in τ_t^*, every time a new shortest path is selected at most two new paths are generated[1].

Therefore, the overall complexity of the algorithm becomes:

$$O(|L| + K) \tag{21}$$

It should be noted that $|L|$ is due to the initial operations that precede the sequential generation of the states, this is what could be called the algorithm 'overhead'.

As the cost matrix is upper triangular the maximum $|L|$ is $(|V|^2 - |V|)/2$, with $|V| = 2 + \sum_{i=1}^{n}(N_i - 1) < 2 + (N-1)n$, where N_i is $d(i) + 1$ and N is the $\max_i N_i$,

$$O(|L| + K) \leq O([nN]^2 + K) \tag{22}$$

3.2. Comparing with the Yang and Kubat algorithm

Yang and Kubat [5] proposed a state enumeration algorithm with multi-mode components where the state enumeration problem for a given state coverage probability is transformed into a tree search problem. This algorithm, the most efficient so far, was shown to have a complexity $O(K\sum_{i=1}^{n} N_i) \leq O(KnN)$, where K is the number of states which had to be generated in order to attain a given coverage probability; also n and N have the same meaning as in this text. Therefore, it may be concluded:

- if $K > [nN]^2$ the presented algorithm has an upper bound complexity of $O(K)$, which is much lower than $O(KnN)$. The situation $K > [nN]^2$ will occur for small and medium size problems, because if nN is high, lets say greater than

[1] This can be deduced from the MPS algorithm with this particular τ_t^*.

1000 then $1000^2 = 1E6$, then probably it will be unfeasible to consider such a significant number of states.

- If $K < [nN]^2$ the proposed algorithm will have an upper bound complexity of $O([nN]^2)$ which is smaller than $O(KnN)$, for $K > Nn$.
  This situation, $K < [nN]^2$, will occur only for large problems and in that case having $K > Nn$, will be the most common in reliability studies! In fact, taking $K = (N-1)n + 1$ would result in considering the most probable state and a number of states equal to the number of different states S_k of cardinality 1.
  Therefore, considering that for large problems usually $nN < K < (nN)^2$, the complexity of the proposed algorithm will be lower than the one of the Yang and Kubat approach.

3.3. Complexity versus memory requirements

The implementation of the MPS algorithm [8] the complexity of which was presented in Section 3.2, uses an address calculation method [9] for orderly storing the candidate paths. This method is extremely efficient, when the arc costs are integers, which is not the case of our cost matrix.

The computational results, in Section 4, refer to an implementation where a binary heap [11] was used for orderly storing the candidate paths. This particular implementation of the MPS algorithm has complexity (see Appendix A):

$$O(|L| + K \log_2 K) \tag{23}$$

This complexity is still lower then the one achieved by the algorithm by Yang and Kubat as long as $\log_2 K$ is smaller than Nn, which will be true for K of significant size when compared to the total number of states of single failure in the network.

Therefore, we may conclude that regardless of the implementation our algorithm presents lower complexity.

As for memory requirements, the proposed algorithm stores at most $2K + 1$ paths and therefore has a memory complexity of $O(K)$, because each path can be stored using a record of fixed size (see Appendix D), which does not depend on n or N.

The memory requirements of the algorithm of Yang and Kubat [5] are not so easily obtained. From the calculations in Appendix B, a lower bound for the number of nodes needed to represent K states, is given by $n - h + K\sum_{j=0}^{h} 1/N^j$. Considering only the first three terms of the sum we have $n - h + K + K(1/N + 1/N^2)$. Considering that any node (except the K leaves) has an associated array of size N_i (for storing the weights of the heaviest leaves of its sub-trees [5]), then an approximate lower bound for the memory requirement of this algorithm is $K(2 + 1/N) + N(n - h)$. Therefore, a lower bound for memory complexity is $O(K + nN)$.

The *lower* bound for the Yang and Kubat algorithm is therefore similar to the *upper* bound of the proposed algorithm, and computional experiments indeed suggest that the memory requirements for the Yang and Kubat algorithm are far from the best case.

4. Experimental results

In the graphics that follow 'YK' will be used for the Yang and Kubat algorithm and 'MM' for the proposed algorithm. The state probabilities were randomly generated, assuring that the operational state probability of the components was always greater than to 0.99 and 0.999, for $n < 500$ and for $n \geq 500$, respectively; the number of states per component was also randomly generated in $\{2,3,\ldots,N\}$. The CPU times are presented as a function of K (number of selected network states) in Fig. 1 and as a function of n (number of network components) in Figs. 2 and 3. A Pentium III at 500 MHz with 256 MB of memory, running Linux with 128 MB for swapping, was used.

For networks with a small number of components ($n = 50, 100, 200$) the performance of MM is superior to YK; for larger problems such as networks with 500 components, only when the number of selected states is relatively very low (in the example $K = 50, 100$, in a 500 element network with at most five different modes) does MM perform worse than YK. Nevertheless, this result does not compromise the use of MM in practical cases because in any reliability study the number of network states will have to be larger than n,

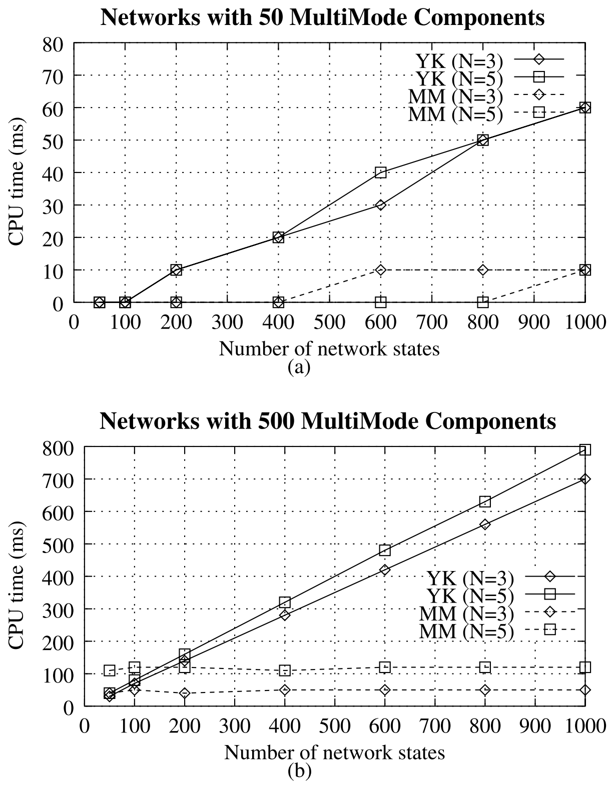
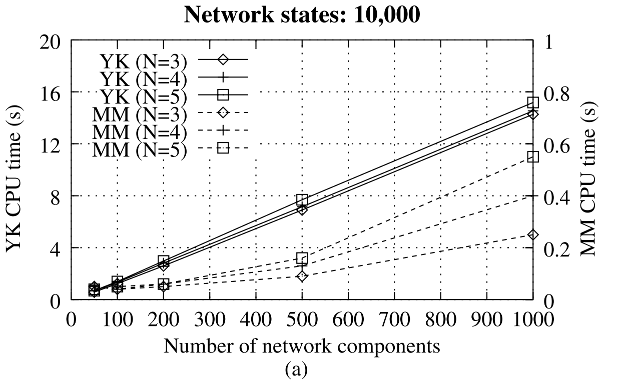


Fig. 1. Comparison of the algorithms for $N = 3, 5$ and (a) 50 (b) 500 network elements; the curves in (a) for YK($N = 3$) and YK($N = 5$) overlap in most cases; CPU time of *zero* in the graphics means exactly CPU time less than 10 ms (available accuracy).



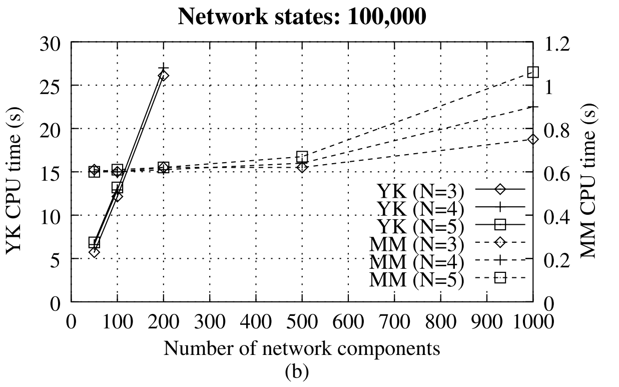
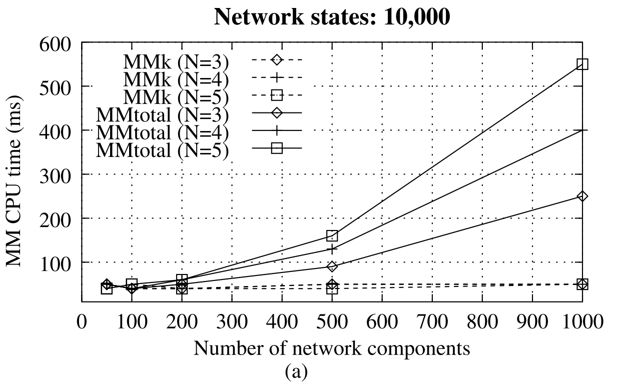


Fig. 2. Comparison of the algorithms for $N = 3, 4, 5$, (a) 10,000 states and (b) 100,000.



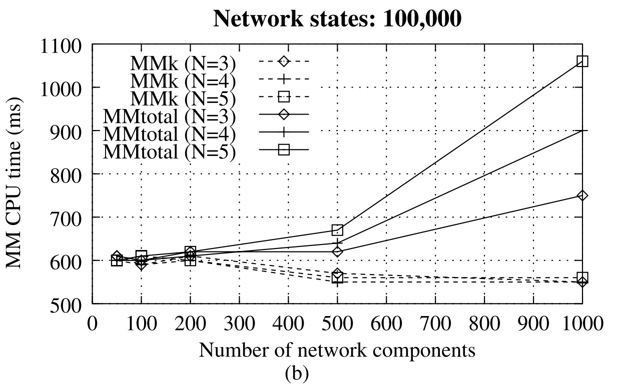


Fig. 3. Total CPU time (MM-total) versus part of that time which was used for state enumeration (MM-k), after the initial overhead, for$N = 3, 4, 5$, (a) 10,000 states (b) 100,000 states.

and indeed greater than $n(N-1)$ which is the number of states of single failure—and for those conditions the proposed algorithm is significantly more efficient.

The almost flat line in Fig. 1(a) and (b) for MM is due to the fact that, if the number of states K is not very large compared with nN, most of the CPU time is overhead time, and therefore, the time used to actually select the states in almost irrelevant. In Fig. 3(a), for $n = 1000$ this condition still holds, but for $n = 50, 100, 200, 500$ the situation is different. The situation reverses in Fig. 3(a) and (b) for $n = 50, 100, 200$, where the time after the overhead becomes dominant; still in Fig. 3(a) for $n = 500$ and $N = 3$ the overhead time is approximately equal to the remaining time for state selection, therefore they have identical importance.

Please note the different scales for CPU time in Fig. 2(a) and (b) for both algorithms. For example for $n = 1000$, $N = 5$ ($nN = 5000$), the results show a CPU time of 15.8 s for YK and 0.55 s for MM (of which 0.5 s are due to the initial overhead cost). It is precisely in large problems that the proposed algorithm performs best when compared with the Yang and Kubat approach, as can be seen from Fig. 2(a) and (b). In fact for $n = 1000$ the tree height will be $n + 1$ in Yang and Kubat algorithm, but in the MM algorithm, apart from the initial cost (0.5 s for $N = 5$) the CPU time for MM grows linearly with n, as well as memory.

In Fig. 2(b) there are no CPU times for YK for $n = 500, 1000$ because the Yang and Kubat algorithm uses all available memory, then starts to use disk space as memory, finally uses all of the swap disk available and then terminates due to lack of resources, before obtaining the desired 100,000 states. In Fig. 2(b), the CPU values for $n = 100$ are still without swapping but for $n = 200$, swapping already occurs. These results confirm what was expected from the analysis of the memory requirements for the worst case and the best case for the MM and YK algorithms respectively, which were shown to be close.

In MM the initial cost, which grows with the problem dimension and wherein lies the most significant part of the CPU effort for larger problems ($n = 500, 1000$) and small number of network states, can be seen in Fig. 3(a) (where MM-total represents the total CPU time and MM-k represents the part of that CPU time that is used for enumerating the $K = 10000, 100000$ states, after the initial overhead). This stems from the fact that in those problems the complexity of the proposed algorithm is established by the algorithm 'overhead': the effort that precedes the first state selection, regardless of the number of selected states K, as long as $K < (nN)^2$ (according to the complexity calculations). Nevertheless as the number of states becomes more significant, the cost of the state generation surpasses the initial cost, as shown in Fig. 3(b) for $n = 50, 100, 200, 500$, and this makes the extreme efficiency of MM clearer when compared with YK, as can be seen in Fig. 2(b).

5. Conclusions

In the context of performability analysis of telecommunications networks it is necessary to select the states to be analysed, thus requiring an efficient algorithm for generating the network failure states by decreasing probability until a certain probabilistic coverage of the state space is attained.

We have presented a new algorithm for the purpose of enumerating, by decreasing probability order the most probable states in a network with multi-mode components. The algorithm efficiency results from using a simplified version of an already extremely efficient *K*-shortest paths algorithm. For this purpose an adequate graph and cost matrix have to be built, so that each path between two special nodes represents a network state.

The proposed algorithm presents lower complexity than the most efficient algorithm known in the literature [5] when the number of selected states is of practical interest. Also it uses much less memory which may be an important factor in the context of a reliability analysis tool for problems of great dimension.

Finally some computational results were presented that showed the signifcant efficiency improvement which can be achieved by using this algorithm instead of the Yang and Kubat [5] approach.

Acknowledgements

The authors are very grateful to Professor Ernesto Martins, Marta Pascoal and José Luis Santos for their explanations concerning the MPS algorithm and its implementation. Célia Reis implemented both algorithms and collected some of the data used in the figures of this paper.

Appendix A. Complexity calculations

Let $|X_k|$ be the size of the set of candidate paths before the removal of the *k*th shortest path. In the worst case, two insertions will take place after the selection (and removal) of every selected path, therefore, $|X_{k+1}| = |X_k| + 1$. So, having in mind that the insertion of a node or the removal of the root of a binary heap has complexity $O(\log_2 b)$ where b is the heap size, the cost of maintaining a heap of shortest paths is:

$$1 + \sum_{k=1}^{K-1} (2\log_2|X_k| + \log_2|X_{k+1}|) + \log_2|X_K| \tag{A.1}$$

But $|X_1| = 1$ (initially the set of candidate paths is just the shortest path from s to t in τ_t^*), $|X_2| = 2, \ldots, |Xk| = k$, and the previous expression can be rewritten:

$$1 + \sum_{k=1}^{K-1} (2\log_2 k + \log_2 k + 1) + \log_2 K < 3K\log_2 K \tag{A.2}$$

Table C.1
Probabilities of component modes

i	$p(i,0)$	$p(i,1)$	$p(i,2)$
1	0.7	0.3	
2	0.5	0.2	0.3
3	0.8	0.2	

Therefore, the cost of the algorithm is now $O(|L| + K\log_2 K)$.

Appendix B. Memory requirements

The calculation of the memory requirements of the Yang and Kubat [5] algorithm is not simple.

The Yang and Kubat algorithm builds a tree of height $n + 1$, where every path from the root to each of the leaves corresponds to a network state. The number of nodes in that tree, necessary for calculating the K most probable states, depends on the values of the $p(i,j)$ for every component i.

A lower bound on the number of nodes can nevertheless be obtained. Let's consider that the tree of the K most probable states has a subtree of height $1 + \log_N K$ (that is the most dense sub-tree with K leaves) hanging from an arm with $n + 1 - h$ nodes, where $h = \lceil \log_N K \rceil$ ($\lceil x \rceil$ represents the smallest integer greater or equal to x).

So the total number of nodes is $n - h + K\sum_{i=0}^{h} 1/N^i$. It should be noted that such a tree is a very unlikely structure.

Appendix C. Example

An illustrative example of the calculation of R is presented in Table C.2 for three components with two and three possible modes (with probabilities given in Table C.1, identical to the ones used in Ref. [5]). Table C.3 and Fig. C.1 (where costs have been rounded) present the corresponding cost matrix and graph, respectively. From the graph in Fig. C.1 and the cost matrix in Table C.3 it is easily seen that τ_t^* will only have arcs of null cost: the arcs in the last column of the cost matrix, $(i, t = 5)$ with $i = 0, 1, 2, 3, 4$.

In this example, the number of disconnected modes, $d(i)$, of each component i is: $d(1) = 1$, $d(2) = 2$ and $d(3) = 1$; therefore, $d_T(1) = 1$, $d_T(2) = 3$, $d_T(3) = 4$ and $d_T(i-1) < r \le d_T(i)$ means that index r refers to the jth ($j = r - d_T(i-1)$) disconnected mode of component i (with

Table C.2
Calculation of R according to values in Table C.1

(i,j)	r	$R(r)$	$-\ln R(r)$
(1,1)	1	0.3/0.7	0.84730
(2,1)	2	0.2/0.5	0.91630
(2,2)	3	0.3/0.5	0.510826
(3,1)	4	0.2/0.8	1.38629

Table C.3
Cost matrix

	$s=0$	1	2	3	4	$t=5$
$s=0$	∞	0.8473	0.9163	0.510826	1.38629	0
1	∞	∞	0.9163	0.510826	1.38629	0
2	∞	∞	∞	∞	1.38629	0
3	∞	∞	∞	∞	1.38629	0
4	∞	∞	∞	∞	∞	0

$d(0) = d_T(0) = 0$). Note that, in Table C.3, $c_{23} = \infty$ because $\mathrm{id_e}(2) = \mathrm{id_e}(3)$.

Appendix D. Brief revision of MPS algorithm

Each time a path p is chosen from a set of candidate paths, X, new paths may be added to X. In the context of the MPS [8] algorithm the node v_k of path p, from which a new candidate path is generated is *the deviation node* of that new path (which coincides with p up to v_k); in a path the link the tail of which is the deviation node, is called the *deviation arc* of that path. By definition s is the deviation node of p_1 (the shortest path from s to t).

Let $a = (i,j) \in L$, then nodes i and j are the *tail* and *head* of arc a, respectively.

The *concatenation* of path p, from i to j, with path q, from j to l, is the path $p\Diamond q$, from i to l, which coincides with p from i to j and with q from j to l.

Let τ_t designate a tree where there is a unique path from any node i to t (tree rooted at t) and $\pi_i(\tau_t)$ denote the cost of the path p, from i to t, in τ_t; the *reduced cost* $\bar{c}_{ij}$ of arc $(i,j) \in L$ associated with τ_t is $\bar{c}_{ij} = \pi_j(\tau_t) - \pi_i(\tau_t) + c_{ij}$.

Let τ_t^* be the tree of the shortest paths from all nodes to t and $p^*_{v_jt}$ the shortest path from v_j to t in τ_t^*. The sub-path from v_i to t in p_k is represented by $p^k_{v_it}$, and the sub-path from s to v_i by $p^k_{sv_i}$.

Let the set of arcs L of (L, V) be written in terms of $A(k)$, the set of arcs the tail node of which is $v \in V = \{1, 2, \dots, n\}$, i.e. $L = A(1) \cup A(2) \cup \dots \cup A(n)$ such that $A(i) \cap A(j) = \emptyset$ for any $i \neq j$ $(i,j \in V)$. Let $a'_k \in A(\xi)$ and $a'_l \in A(\theta)$. Assuming that $\xi \neq \theta$ an order relation '$<$' is defined for the arcs such that $a'_k < a'_l$ iff $\xi < \theta$. Moreover if $\xi = \theta$ then $a'_k < a'_l$ if $\bar{c}(a'_k) \leq \bar{c}(a'_l)$.

This means that the set A is sorted in such a way that for any two arcs $(k,j), (i,l) \in L$, $(k,j) < (i,l)$ if $k < i$ or ($k = i$ and $\bar{c}_{ij} \leq \bar{c}_{il}$). The resulting set $L = \{a_1, a_2, \dots, a_m\}$ is said to be in the *sorted forward star form*.

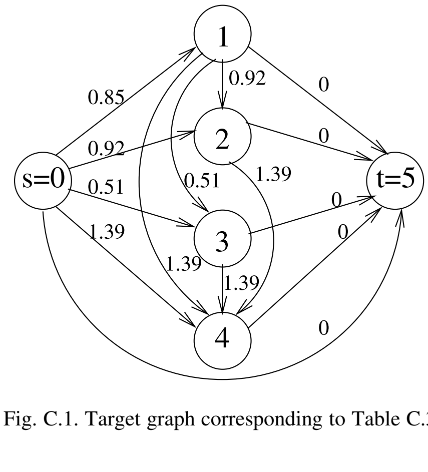


Fig. C.1. Target graph corresponding to Table C.3 with arc costs rounded.

MPS Algorithm (adapted).

```
1. Input: the representation of the graph $(V, L)$, arc costs $c_{ij}$, $s$ and $t$.
2. Compute $\tau_t^*$.
3. Calculate the reduced cost $\bar{c}_{ij}$ for every $(i,j) \in L$.
4. Rearrange the arcs of $(V, L)$ in the sorted forward star form (for the computed $\bar{c}_{ij}$).
5. $p_1 \leftarrow$ shortest path from $s$ to $t$ ($p_1 \in \tau_t^*$)
6. $X \leftarrow \{p_1\}$
7. $k \leftarrow 0$
8. While $((k < K) \wedge (X \neq \emptyset))$ Do
     (a) $k \leftarrow k + 1$
     (b) $p_k \leftarrow$ shortest path in $X$
     (c) $X \leftarrow X - \{p_k\}$
     (d) Let $v_i$ be the deviation node of $p_k$
     (e) Repeat
       (i) $a_h \leftarrow$ the arc of $p_k$ the tail of which is $v_i$
       (ii) If ($v_i$ is the tail of $a_{h+1}$) Then
           – $v_j \leftarrow$ head of $a_{h+1}$
           – $X \leftarrow X \cup \{p^k_{sv_i} \Diamond \langle v_i, a_{h+1}, v_j \rangle, p^*_{v_jt}\}$
       EndIf
       (iii) $v_i \leftarrow$ following node in $p_k$
     Until $(v_i = t)$
9. EndWhileDo
```

Each path could be represented by the following information: cost of the path, deviation arc of the path and the order of the path it deviates from. The paths are stored in a pseudo-tree as described in Ref. [8].

References

[1] Meyer JF. Performability: a retrospective and some pointers to the future. Perform. Eval. 1992;14/(3,4):139–56.
[2] Li VOK, Silvester JA. Performance analysis of networks with unreliable components. IEEE Trans. Commun. 1984;32(10):1105–10.
[3] Lam YF, Li VOK. An improved algorithm for performance analysis of networks with unreliable components. IEEE Trans. Commun. 1986;34(5):496–7.
[4] Yang C-L, Kubat P. An algorithm for network reliability bounds. ORSA J. Comput. 1990;2(2):336–45.
[5] Yang C-L, Kubat P. Efficient computation of most probable states for communication networks with multimode components. IEEE Trans. Commun. 1989;37(5):535–8.
[6] Gomes T, Craveirinha J. An algorithm for the sequential generation of states in a failure prone communication network. IEE Proc. Commun. 1998;145(2):73–9.

[7] Chiou S-N, Li VOK. Reliability analysis of a communication network with multimode components. IEEE J. Select. Areas Commun. 1986;4(7):1156–61.

[8] Martins E, Pascoal M, Santos J. Deviation algorithms for ranking shortest paths. Int. J. Found. Comput. Sci. 1999;10(3):247–63.

[9] Dial R, Glover F, Karney D, Klingman D. A computational analysis of alternative algorithms and labeling techniques for finding shortest path trees. Networks 1979;9(3):215–348.

[10] Eppstein D. Finding the *k* shortest paths. SIAM J. Comput. 1998;28(2):652–73.

[11] Sedgewick R. Algorithms in C++. Reading, MA: Addison-Wesley, 1992.