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

by

Teresa Gomes

José Craveirinha

Lúcia Martins

INESC – Coimbra

Research Report ET-N7
July 2000

Work supported by FCT, project PRAXIS/P/EEI/13219/1998, *Um estudo sobre encaminhamento dinâmico multi-objectivo e dependente do estado em redes multi-serviço*

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

Teresa Gomes
José Craveirinha
Lúcia Martins

Departamento de Engenharia Electrotécnica,
Pólo II da Universidade de Coimbra,
Pinhal de Marrocos, 3030 COIMBRA, Portugal.

INESC-Coimbra, Rua Antero de Quental 199,
3000-033 COIMBRA. Portugal.

e-mail: teresa@dee.uc.pt, jcrav@dee.uc.pt, lucia@dee.uc.pt

26th July 2000

Abstract

In this work a new algorithm for the sequential generation of failure states in a network with multi-mode components is proposed. The presented algorithm 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, we obtain an algorithm with lower complexity than the best algorithm in the litterature for solving this problem.

Results will be presented for used CPU time for both algorithms.

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 eg. Meyer [8] leading to the concept of "performability". In this type of approach the possible

network states are enumerated alongside their steady state probabilities and an overall performability measure is calculated as the network performance averaged over all network state probabilities. 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 & Silvester [6] 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 & Li [5] and later by Yang & Kubat [11], taking as a basis an algorithm [10] 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 Teresa & Craveirinha [4]. For networks with multi-mode components (the focus of this paper) an algorithm, generalizing the Lam & Li [5] approach was introduced by Chiou & Li [1]. The performance degradation of each component is here characterized by up to N different modes (or states) and the algorithm enumerates network states in order of decreasing probability. For a network with n components the complexity of the algorithm by Chiou & Li is $\mathcal{O}(n^2Nm + Nm\log m)$ where m is the number of enumerated states. This algorithm also requires the selected value of m to be predermined. Another more efficient algorithm was proposed by Yang & Kubat [10] which was shown to have an upper limit complexity $\mathcal{O}(nN|\Omega_\varepsilon|)$ when used for generating the most probables states corresponding to the minimal set Ω_ε which satisfies the requirement of a coverage probability of the state space not less than $1-\varepsilon$. It is at least n times faster than the one by Chiou and Li [1]. Other 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 multimode components. This algorithm is based on the transformation of the enumeration problem into a K-shortest path problem. The major advantage of the proposed algorithm results its lower complexity and therefore lower running CPU time and also from the quite significant reduction in memory requirement as compared with the Yang & Kubat [10] algorithm. These two characteristics are an important criterion of performance comparison, particularly decise in network of great dimension as is usually is the case of communication networks, such as multiexchange networks.

The paper is organised as follows. Section 2 introduces basic concepts of the underlying model and previous results necessary for justifying the algorithm. The formalisa-

tion of the algorithm, its complexity analysis and memory requirements are presented in section 3, some computational results comparing, in terms of CPU time the Yang & Kubat [10] 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

Le us consider a network with n components where each component can be in $j = 0, 1, 2, \ldots, d(i)$ different modes, corresponding to different operational, partially operational or inoperational conditions of the component. It is assumed that component failures are statiscly 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 $op(i) = p(i,j_0)$ be the probability of the component being fully operational, then the probability of being in any inoperability sate is $ip(i) = 1 - op(i)$. In pratical situations it only interest $1/2 \leq op(i) < 1$, $p(i,j_0) \geq 1/2$. Nevertheless in order to mantain the algorithm as general as possible the follwing definitions are introduced, inspired in the definitions in [6]. A component is said to be connected if it is in its most probable state; otherwise 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 $op(i) \geq ip(i)$ then $ip(i) = op(i)$. If $op(i) < ip(i)$ the $p(i) = p(i,j_m)$ (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 $q(i,j)$:

$$q(i,j) = p_e(i,j'), \quad \text{with} \quad \begin{array}{l} i = 1, 2, \ldots, n \\ j' = 0, 1, \ldots, j_m - 1, j_m + 1, \ldots, d(i); \\ j = \left\{ \begin{array}{ll} j'+1 & if \quad j' < j_m \\ j' & if \quad j' > j_m \end{array} \right. \end{array} \tag{1}$$

Let S_k designate the states of the network. Then similarly to [6]

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

where

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

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 $R(v)$ is associated with a given disconnected mode of some component of the network:

$$R(r) = \frac{q(i,j)}{p(i)}, \quad \begin{array}{ll} i = 1,2,\ldots,n; & j = 1,2,\ldots,d(i); \\ d_T(i) = d(i-1) + d(i); & r = d_T(i-1)+1,\ldots,d_T(i) \end{array} \tag{4}$$

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

A function id is defined, that transforms the index $v = \{1,2,\ldots,d_T(n)\}$ of R into the pair (i,j) which identifies the $j-$th disconnected state of component i, according to (1) and (4). Other function id_ϵ 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 equations (1) and (4).

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

$$\forall e_v, e_u \in S_k : id_\epsilon(e_v) \neq id_\epsilon(e_u) \tag{5}$$

and from (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 shortest path algorithms 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 signal "-" guarantees that only positive values are considered.

A graph is then considered where paths are originated at a fictitious node $s = 0$ and terminated 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 analyzed hereafter.

Let $\mathcal{V} = \{s, v_1, v_2, \ldots, v_{t-1}, t\}$ be the node set of of a directed graph and $\mathcal{L}$ the arc set, composed of ordered pairs of elements in $\mathcal{V}$. The k-th path generated by the algorithm in this graph will be specified by the sequence $p_k = \langle s, (s,v_1), v_1, \ldots, (v_w, t), t\rangle$.

The corresponding state is $S_k = \{v_1, v_2, \ldots, v_w\}$, where the auxiliary nodes s,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 (7)

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

This implies that

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

and:

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

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

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 satisfaying relations (4), (11) and (12)

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

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

$$c_{ak} = \infty, \quad if \quad 0 < a < k \wedge id_\epsilon(a) = id_e(k) \tag{15}$$

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

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

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 $p_1 = \langle s, (st), t \rangle$, 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 can not exist. Equation (14) prevents the generation o paths including repeated nodes and the generation of different paths, formed by the same set of nodes, placed in different order. Finally equation (17) implies that adding node t to a path does not have any cost, according to (12). It should be pointed out that the obtained matrix is acyclic, that is no path can be constructed with identical orignal and terminal nodes. Therefore K-shortest

paths algorithms will always obtain loopless paths, when applied to a graph whose arcs' cost are given by (13)-(17).

An illustrative example of the calculation of R is presented int Table 1 for 3 components with 2 and 3 possible modes. Table 2 presents the corresponding matrix.

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

(a)

(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

(b)

Table 1: (a) Probabilities of components' modes and (b) calculation of R.

∞	0.84730	0.91630	0.510826	1.38629	0
∞	∞	0.91630	0.510826	1.38629	0
∞	∞	∞	∞	1.38629	0
∞	∞	∞	∞	1.38629	0
∞	∞	∞	∞	∞	0

Table 2: Cost matrix

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 above defined two auxiliary propositions are now presented.

Proposition 2.1 *Any 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 equations (13)-(17) and the arcs with ∞ cost deleted, then $s < v_1 < v_2 < \ldots < v_w < t$.

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

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

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

then $id_e(v_a) \neq id_e(v_b)$, $\forall v_a, v_b \in p$.

Here it is conventioned that $id_e(s) = 0$ *and* $id_e(t) = n + 1$

Proof: *Let us assume* $v_i < v_j$, $v_i, v_j \in p$.

Let p':

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

If $(v_i, v_j) \in p'$ *then from (15),* $id_e(v_i) \neq id_e(v_j)$, *since otherwise the arc would have* ∞ *cost. If* $(v_i, v_j) \notin p'$ *and* $(v_i, v_j) \in p$ *then* $v_i = s = 0$ *or* $v_j = t$ *and the proposition is obviously verified. If* $(v_i, v_j) \notin p'$ *and* $(v_i, v_j) \notin p$ *then, it is possible to construct a sub-path* p^* *from* v_i *to* v_j, *in* p:

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

such that $v_i < v_i' < \ldots v_j' < v_j$ *and from equations (4) and (15) the arcs in* p^* *only may exist if the extreme nodes have different values of* id_e, *which concludes the proof.*

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_v, when added to $-\ln P(S_1)$ gives the values $-\ln P(S_k)$. For calculating the K-shortest paths we will use algorithm MPS in [7], which is, to best of our knowledge, the most efficient algorithm available in the literature.

The variable P_c and P_d represent the calculated and the desired probability coverage, repectively.

Algorithm 3.1 (State generation in a multimode component network)

1. *Input:* $p_e(i, j)$, $i = 1, 2, \ldots, n$, $j = 0, 1, 2, \ldots, d(i)$. *and* P_d, *the desired coverage probability of the state space and assume without loss of generality:* $p_e(i, 0) = \max_{j=0,\ldots,d(i)} p_e(i, j)$.

2. *Calculate* $P(S_1)$, $P(S_1) = \prod_{i=1}^{n} p_e(i, 0)$.

3. *Construct a vector* R, *such that:*

$$R(r) = \frac{p_e(i, j)}{p_e(i, 0)}, \quad \begin{array}{lll} i = 1, 2, \ldots, n; & & j = 1, 2, \ldots, d(i); \\ d_T(i) = dT(i-1) + d(i); & & r = d_T(i-1) + 1, \ldots, d_T(i); \end{array}$$

with $d(0) = 0$ *and* $d_T(0) = 0$.

4. *Define the graph the arcs of which have associated costs* $[c_{ak}]$ *given by (13)-(17)*

5. *Construct the shortest tree of all nodes to* t. *This is trivial because such tree is formed by the paths* $< v, (v,t), t >$ *of cost 0, with* $v = 0, 1, 2, ...t-1$.

6. *Obtain a representation of the graph in the ordered forward star form (see details in Dial & et al. [2])*

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 used K-shortest path algorithm MPS is given in [7]:

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

where each of the parcels is due to:

- $|\mathcal{L}| + |\mathcal{V}| \log_2 |\mathcal{V}|$: obtaining the tree of shortest paths to t, τ_t^*;
- $|\mathcal{L}|$: calculation of reduced costs in the arcs[3, 7], necessary for applying MPS algorithm;
- $|\mathcal{L}| \log |\mathcal{V}|$: storing the arcs in the sorted forward star form;
- $K|\mathcal{V}|$: cost of obtaining the K shortest paths in τ_K, after performing the previous operations. This results from the fact that, in MPS, each time a shortest path is selected, at most $|\mathcal{V}|$ new paths have to be generated.

This gives rise to a simplified expression for the complexity of the form [7]:

$$\mathcal{O}\left(|\mathcal{L}|\log|\mathcal{V}| + K|\mathcal{V}|\right) \tag{19}$$

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 $\mathcal{O}\left(|\mathcal{V}|\right)$.
- Also in the context of the proposed algorithm it is not necessary to calculate reduced costs in the arcs (reported to the shortest path tree rooted at t) since they are equal to the arcs 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 $\mathcal{O}\left(|\mathcal{V}|\log_2|\mathcal{V}|\right)$ [9];
  - 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:

  $$\mathcal{O}\left(|\mathcal{L}| + \mathcal{V}|\log_2|\mathcal{V}|\right) = O\left(|\mathcal{L}|\right)$$

  assuming $|\mathcal{L}| \ll |\mathcal{V}|^2$.
- Due to the fact that all nodes in $\mathcal{V}$ 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:

$$\mathcal{O}(|\mathcal{L}| + K) \tag{20}$$

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

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

Therefore, if the number of selected states is sufficiently large, the complexity of the algorithm will depend solely on the number of selected states, K.

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

$$\mathcal{O}\left(|\mathcal{L}| + K\right) \leq \mathcal{O}\left([nN]^2 + K\right) \tag{21}$$

or,

$$\mathcal{O}\left(|\mathcal{L}| + K\right) \leq \begin{cases} \mathcal{O}\left(K\right) & \text{if } K > [nN]^2 \\ \mathcal{O}\left([nN]^2\right) & \text{if } K < [nN]^2 \end{cases} \tag{22}$$

3.2 Comparing with the Yang and Kubat algorithm

Yang & Kubat [10] 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 show to have a complexity $\mathcal{O}\left(K\sum_{i=1}^{n} N_i\right) \leq \mathcal{O}\left(KnN\right)$, 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 $\mathcal{O}\left(K\right)$, which is much lower than $\mathcal{O}\left(KnN\right)$

  The situation $K > [nN]^2$ will occur for small and medium size problems, because is nN is high, lets say greater than 1000 then $1000^2 = 1E6$, and 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 $\mathcal{O}\left([nN]^2\right)$ which is smaller than $\mathcal{O}\left(KnN\right)$, 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 comom 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 differente states, S_k of cardinality 1.

  Therefore usually, for large problems $nN < K < (nN)^2$, and the complexity of the proposed algorithm will be apparently only slightly inferior to the Yang & Kubat approach.

3.3 Complexity versus memory requirements

The implementation of the MPS algorithm [7], the complexity of which was presented in the previous sub-section, uses most likely an address calculation method [2], for

orderly storing the candidate paths. This method is extremelly efficient, when the arcs' costs are integers.

The constructed cost matrix is made of real numbers obtained from using logarithms over probability values. Many of these real numbers will differ only their fractional parts, and the effciency gained by using address calculation method [2], will be lost.

The computional results, in the next section, refer to an implementation where a binary heap [9] was used for orderly storing the candidate paths. This paticular implementation of the MPS algorithm has complexity (see appendix A for detailled calculations):

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

This complexity is still lower then the one achieved by the algorithm by Yang & Kubat as long as $\log_2 K$ is smaller than Nn, which will be true for K of significant value 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 $\mathcal{O}(K)$, because each path can be stored using a record of fixed size, which does not depend on n or N.

The memory requirements of the algorithm of Yang & Kubat [10] is not so easily obtained. From the calculations in the appendix, a lower bound for the number of nodes needed, to represent K states, is given by $n + 1 - h + K \sum_{i=0}^{h} 1/N^i$. Considering only the first three parcels of them sum we have $n+1-h+K+K(1/N+1/N^2)$ Considering that everyone of these nodes (except the K leafs) has associated an array of size N_i (for storing the wheight of the most heavy leaf of its i-th sub-tree), than an approximate lower bound for the memory requirement of this algorithm is $K(2+1/N)+N(n+1-h)$. Therefore a lower bound for memory complexity is $\mathcal{O}(K + Nn)$.

The *lower* bound for the Yang and Kubat algorithm is therefore similar to the *upper* bound of the proposed algorithm, and the computional results will show that indeed the results are for from the best case.

4 Experimental results

In the graphics that follow we will use "YK" for the Yang and Kubat algorithm and "MM" for the proposed algorithm.

The $\mathcal{O}$ symbol is a rather crude operator, and the experimental results will show that althoug the complexity of the two algorithms seems to be close values of K, $k \in [nN, (nN)^2]$, the experimental results show otherwise, as can be seem from figures

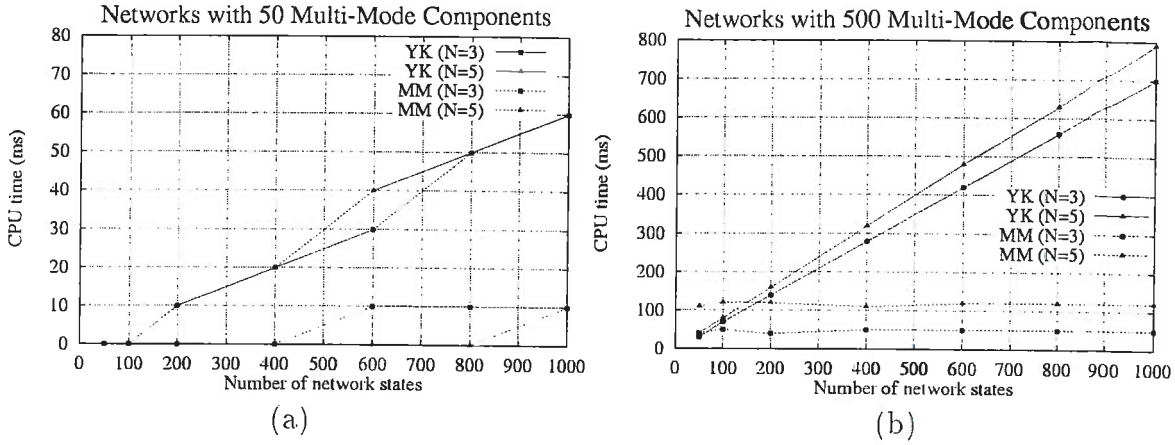

Figure 1: Comparison of the algorithms for $N = 3, 5$ and (a) 50 (b) 500 network elements

1(a) and (b). For neworks 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 5 diferent modes) does MM performs worse than YK. Nevertheless this result does not compromise the use of MM because in any reliability study the number of network states will have to be larger than n, and indeed superior to $n(N-1)$, which is the number of states of single failure – and for those conditions the prosposed algorithm is significantly more efficient.

The almost flat line in figures 1(a) and (b) for MM is due to the fact that if the number of states K is not very large with nN most of the CPU time is overhead time and therefore the time used to actually select the states in almost irrelevant. In figure 3(a), for $n = 1000$ this condition still holds, but for $n = 50, 100, 200, 500$ the situation is different. The situation reverses, as can be seen in figure 3, where the time after the overhead becomes dominant for $n = 50, 100, 200, 500$; still for $n = 1000$ and $N = 5$ the overhead time is aproximately equal to the time remaining time for state state selection, therefore the they have identical importance.

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

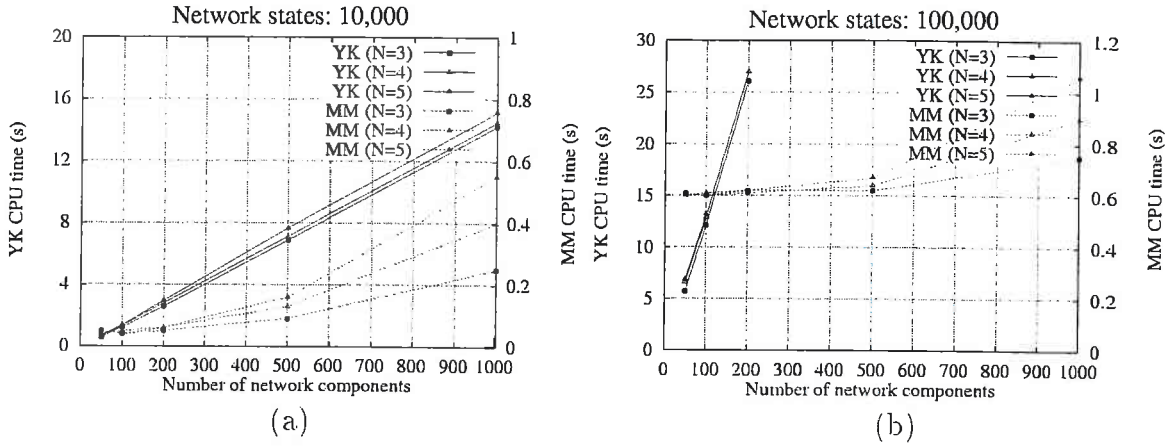

Figure 2: Comparison of the algorithms for $N = 3, 4, 5$, (a) 10,000 states and (b) 100,000

In figure 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 do 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 figure 2(b) the CPU values for $n = 100$ are still without swapping but for $n = 200$, swapping already occurs. This results confirm what was expected from the analys the memory requirements for the worst case and best case for MM and YK algorithms respectively, which were shown to be close.

In MM the initial cost, which grows with the problem dimension, and where lies the most part of the CPU effort, for larger problems (n=200,500,1000) and small number of paths, can be seen in figures 3(a) and 3(b), where (MM-total represents the total CPU time and MM-k represents the part of that CPU time which used for enumerating the $K = 10000, 100000$ states, after the initial overhead). This steems from the fact that in those problems the complexity of the proposed algorithm is established by the algorithm "overhead": the effort that preceeds the first path selection, regardeless of the number of selected paths K, as long as $K < (nN)^2$ (according to the complexity calculations). Nevertheless as the number of paths becomes more significant, the cost of the paths surpassses the initial cost and makes more clear the extreme effciency of MM.

5 Conclusions

In the context of performability analysis of telecomunications networks it is necessary to select the states to analyse, and a commomly accepted criterion is the selection of

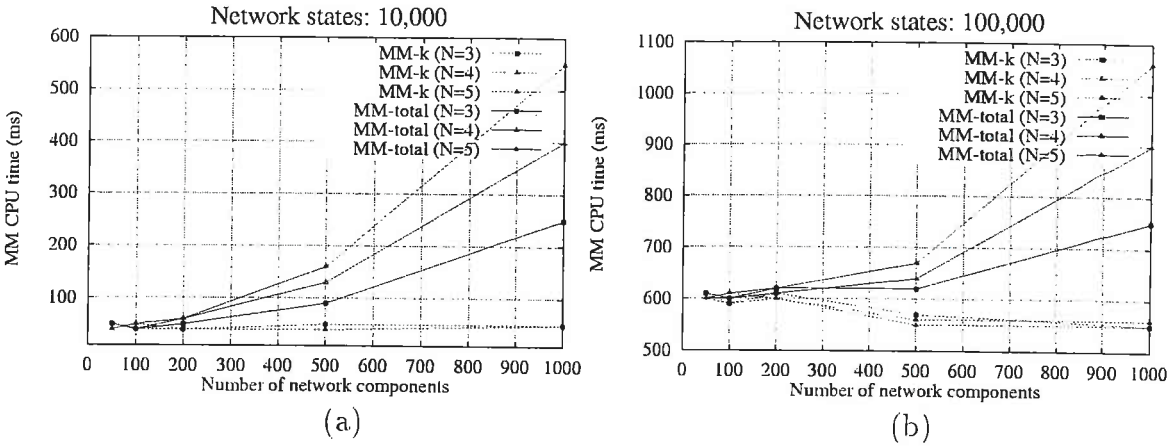

Figure 3: Total CPU time (MM-total) versus part of that time which was used for state enumeration (MM-k), after the initial overhead, for $N = 3, 4, 5$, (a) 10,000 states (b) 100,000 states

states by decreasing probability until a certain probabilistic coverage of the state space is attained.

We present a new algorithm for the purpose of enumerating, by decreasing order the most probable states ins a network with multi-mode components. This algorithm efficiency results from using a simplified version of an already extremely efficient $K-$shortest paths algorithm. For this purpose an adequate cost matrix as to build, so that each path represents a network state.

This algorithm presents lower complexity than the most efficient one known in the literature [10] when the number o selected states is of pratical interest. Also it uses much less memory, which may be an important factor in the context of a large reliability tool analysis.

Some computational results were presented that showed the signifcant improvement which can be achieved by using this algorithm instead of the Yang & Kubat [10] approach.

Acknowledgements: Célia Reis implemented both algorithms and collected some of the data used in the figures of this report.

A Complexity calculations

Let $|X_k|$ be the size of the set of candidate paths before the removal of the k-th shortes path; let $|X_{k,0}|$ be the size of the set of candidate states after the removal of the k-shortest path; and let $|X_{k,1}|$, $|X_{k,2}|$ be the size of the set of candidate paths after the

first and second insertion made after the removal of the k-shortest path. Then, in the worst case (two insertions for every selected path):

$$|X_{k,0}| = |X_k| - 1 \quad (24)$$

$$|X_{k,1}| = |X_{k,0}| + 1 = |X_k| \quad (25)$$

$$|X_{k,2}| = |X_{k,1}| + 1 = |X_k| + 1 \quad (26)$$

$$|X_{k+1}| = |X_k| + 1 \quad (27)$$

So the cost of maintaining a heap of the shortest paths is:

$$1 + \sum_{k=1}^{K-1} (2\log_2 |X_k| + \log_2 |X_{k+1}|) + \log_2 X_K \quad (28)$$

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

$$1 + \sum_{k=1}^{K-1} (2\log_2 k + log_2 k + 1) + \log_2 K = 1 + 2\log_2 K + 3\sum_{k=1}^{K-1} \log_2 k \quad (29)$$

$$< 3\sum_{k=1}^{K} \log_2 k \quad (30)$$

$$< 3K\log_2 K \quad (31)$$

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

B Memory requirements

The calculation of the memory requirements of the Yang & Kubat [10] algorithm is not simple.

The number of nodes in the tree, of height n 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. Lets consider that the tree of the K most probable states has a subtree of height $\log_N K$ (that is the most densest sub-tree with K leafs), 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 equalt to x).

So the total number of nodes is $n + 1 - h + K\sum_{i=0}^{h} 1/N^i$. It should be noted that such a tree is a very unlikely structure although something of that type might be achieved if all nodes are ordered by decreasing cost of their more probable state the different modes of each componet is also ordered by decreasing probability.

References

[1] S.-N. Chiou and V. O. K. Li. Reliability analysis of a communication network with multimode components. *IEEE Journal on Selected Areas in Communications*, SAC-4(7):1156–1161, October 1986.

[2] R. Dial, F. Glover, D. Karney, and D. Klingman. A computational analysis of alternative algorithms and labeling techniques for finding shortest path trees. *Networks*, 9:275–323, 1979.

[3] D. Eppstein. Finding the k shortest paths. *SIAM Journal on Computing*, 28:652–673, 1998.

[4] T. Gomes and J. Craveirinha. An algorithm for the sequential generation of states in a failure prone communication network. *IEE Proceedings - Communications*, 145(2):73–79, April 1998.

[5] Y. F. Lam and V. O. K. Li. An improved algorithm for performance analysis of networks with unreliable components. *IEEE Transactions On Communications*, Com-34(5):496–497, 1986.

[6] V. O. K. Li and J. A. Silvester. Performance analysis of networks with unreliable components. *IEEE Transactions On Communications*, Com-32(10):1105–1110, 1984.

[7] E. Martins, M. Pascoal, and J. Santos. Deviation algorithms for ranking shortest paths. *International Journal of Foundations of Computer Science*, 10(3):247–263, 1999. http://www.mat.uc.pt/~ marta/research.thml.

[8] J. F. Meyer. Performability: a retrospective and some pointers to the future. *Performance Evaluation*, 14(3,4):139–156, 1992.

[9] R. Sedgewick. *Algorithms in C++*. Addison-Wesley Publishing Company, 1992.

[10] C.-L. Yang and P. Kubat. Efficient computation of most probable states for communication networks with multimode components. *IEEE Transactions on Communications*, 37(5):535–538, May 1989.

[11] C.-L. Yang and P. Kubat. An algorithm for network reliability bounds. *ORSA Journal on Computing*, 2(2):336–345, 1990.