Bounds for point and steady-state availability: an algorithmic approach based on lumpability and stochastic ordering

A. Bušić and J.M. Fourneau

PRiSM, Université de Versailles-Saint-Quentin, 45, Av. des Etats-Unis 78000 Versailles, France

Abstract. Markov chains and rewards have been widely used to evaluate performance, dependability and performability characteristics of computer systems and networks. Despite considerable works, the numerical analysis of Markov chains to obtain transient or steady-state distribution is still a difficult problem when the chain is large or the eigenvalues badly distributed. Thus bounding techniques have been proposed for long to analyze steady-state distribution.

Here, we show how to bound some dependability characteristics such as steady-state and point availability using an algorithmic approach. The bound is based on stochastic comparison of Markov chains but it does not use sample-path arguments. The algorithm builds a lumped Markov chain whose steady-state or transient distributions are upper bounds in the strong stochastic sense of the exact distributions. In this paper, the implementation of algorithm is detailed and we show some numerical results. We also show how we can avoid the generation of the state space and the transition matrix to model chains with more than 10^{10} states.

This work is supported by ACI Sécurité, project Sure-Paths

1 Introduction

The use of Markov chains to model complex system reliability and availability is becoming increasingly common. The definition and generation of large-scale Markov models from high level specifications is relatively easy and efficient in both time and memory requirements. The remaining difficulty is that of actually solving the Markov chain and deriving useful performance characteristics from it.

Consider an irreducible finite continuous-time Markov chain X whose stochastic transition rate matrix is given by Q. Then there exists the steady-state distribution π of the Markov chain X which is the unique solution to the system of equations $\pi Q = 0$. An availability measure is defined by separating the states into two classes, UP states and DOWN states. A state is said to be UP if the system is operational for that state; otherwise it is DOWN. Let U denote the set of UP states. The reliability at time t is defined as the probability that the system has always been in the UP states between 0 and t:

$$R(t) = Pr(X_s \in U, \forall \ \ s \ \in \ [0, t[).$$

The point availability is the probability that the system is operational at time t:

$$PAV(t) = Pr(X_t \in U)$$

and the steady-state availability is the limit, if it exists, of this probability. It can also be defined as the expectation of a reward on the steady state distribution of X: $A = \sum_{i|i \text{ is up}} \pi(i)$. The usual way to compute these quantities is based on the uniformization method. Let δ be an arbitrary positive value and $\lambda = \max_i\{-Q(i,i)\}+\delta$. Let us build the uniformized version of Q by: $P = (Id - Q/\lambda)$. P is a discrete time Markov chain. Let us denote by P_U the block of P associated to transitions between UP states and let π_0 be the initial distribution of X. Using uniformization, we can compute $R(t)$ by:

$$R(t) = \sum_{n=0}^{\infty} e^{-\lambda t} \frac{(\lambda t)^n}{n!} \pi_0 P_U^n 1$$

and, because of the properties of the exponential function, the summation can be truncated. We first compute $N(t,\epsilon)$ which is the minimal value of n such that $\sum_{n=0}^{n} e^{-\lambda t} \frac{(\lambda t)^n}{n!}$ is larger than $1-\epsilon$, and we finally obtain an approximation of $R(t)$ which is a lower bound of the exact value:

$$R(t) \approx \sum_{n=0}^{N(t,\epsilon)} e^{-\lambda t} \frac{(\lambda t)^n}{n!} \pi_0 P_U^n 1.$$

Let 1_U be the indicator function of set U. We get $PAV(t)$ after a similar construction based on uniformization:

$$PAV(t) \approx \sum_{n=0}^{N(t,\epsilon)} e^{-\lambda t} \frac{(\lambda t)^n}{n!} \pi_0 P^n 1_U.$$

As P and Q have the same steady-state distribution, we can use P to compute or bound the availability. Thus, we must compute transient and steady state probability distribution for matrix P. But for many problems these matrices are so huge that this is not even possible to build them or to solve the steady-state or the transient distribution. Thus we must use methods which provide a guarantee on these reliability measures and which are not limited by the size of the state space. Note that we are interested in bounding continuous time CTMC, however, due to the uniformization process, we consider discrete time Markov chain (DTMC).

Bounding methods have always received considerable attention in performance or reliability evaluation. Indeed, the problems we have to solve are often too complex to be analyzed exactly. For instance, the numerical computation of the steady-state distribution of Markov chains is difficult when the chain is large or the eigenvalues badly distributed. The main approach for bounding steady-state availability has been proposed by Muntz and his co-authors [11,9]. The method has been improved by Carrasco [1], Rubino and Mahevas [10]. The theoretical background is based on Courtois's polyhedral results on steady-state equation [3]. However this method only works for bounding steady-state rewards.

Here we present a new method which allows to obtain bounds for transient and steady-state rewards. Our approach is based on stochastic comparison of Markov chains and lumpability. The stochastic comparison provides the guarantee for both transient and steady-state measures while lumpability allows to consider smaller chains which are easier to solve. The theory is based on an algorithmic derivation for sample-path comparison of Markov Chains based on necessary conditions on the transition matrix. This approach restricted on steady-state distribution and rewards has recently been surveyed [6], LIMSUB an algorithm based on lumpability has been proved [8] and a tool has been demonstrated [7]. Here we show how we can extend this theory for transient rewards as point availability and reliability. As the theoretical background is not based on Courtois's results on steady state, the requirements of our method are distinct from the assumptions needed by Muntz's algorithm and its generalizations.

The comparison of Markov chains requires an order on the state space because the order on random variables is defined by means of the set of non-decreasing functions. Thus, we order the states such that the UP states have indices that are smaller than the indices of the DOWN states, then we can define A as: $A = \sum_i \pi(i)r(i)$ where the reward $r(i)$ is a non-increasing function which is equal to 1 for the first states of the chain (the UP states) and 0 for all the other states. We wish to compute a lower bound for A. We shall let this lower bound be denoted by B. Notice that we may restate the problem by computing an upper bound for $1 - A$. This upper bound will be given by $1 - B$. In this case, the reward function is now a non-decreasing function on the state indices already defined. This property is an important requirement of the strong stochastic ordering as we will see in section 2.

Here we consider the modeling of highly available multicomponent systems such as the example studied by Muntz [11] and Carrasco [1]. A typical system consists of several disks, CPUs and controllers. We have two types of failures: soft and hard. The failures may occur in batches and all the failed items compete to be repaired. The system is operational if there is enough CPU, disks and controllers. Clearly, if the number of components is large, the state space is huge and the UP states are relatively rare. Furthermore, if the system is highly available, the UP states concentrate most of the probability distribution. For instance, the system depicted in Fig. 1 has more than $9.0\ 10^{10}$ states and 10^{12} transitions. This is even not possible to generate and store the state space and the transition matrix. Thus we show how we can operate in two phases: the first step consists in designing an ad-hoc algorithm (called LL, Lumpable and Larger) which builds from the specification a lumpable matrix which is larger in the stochastic sense. Of course, we store the lumped matrix instead of the original one. Then, during the second phase, we can apply the new bounding algorithm LMSUB to obtain the final matrix which can be numerically analyzed. As the chain is huge we must derive very efficient algorithms. So we report in section 4 some details about an efficient implementation of our new algorithm LMSUB (Lumpable Monotone Stochastic

Upper Bound). As LMSUB is strongly related to LIMSUB these details can also be used to program a more efficient version of LIMSUB than the description in [8]. LMSUB algorithm is devoted to the study of problems with reducible chains while LIMSUB has several instructions to build an irreducible chain. This is the main difference between these algorithms. LMSUB has been specially developed to study reliability issues which imply chains with absorbing states.

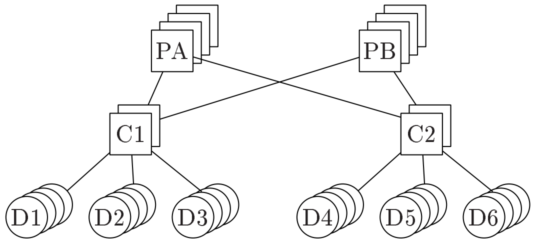

Fig. 1. System studied by Muntz and Carrasco

The paper is organized as follows. In section 2 we present the basic results we need about stochastic bounds, lumpability and algorithmic comparison of Markov chains. Section 3 is devoted to the theoretical aspects of bounding transient rewards. In section 4 we show how we can improve the algorithms to deal with extremely huge state space. Finally, we present some numerical results in section 5 for some typical problems introduced by Muntz and his colleagues [11].

2 Stochastic Bounds and Lumpability

We restrict ourselves to discrete time Markov chains (DTMC) with finite state space $E = \{1, 2, \ldots, n\}$. The strong stochastic ordering ("st-ordering" for short) has been defined by Stoyan[14] by means of the set of non-decreasing functions. For discrete random variables, we use the following algebraic equivalent formulation which is much more convenient:

Definition 1. *If X and Y are random variables taking values on a finite state space $\{1, 2, \ldots, n\}$ and respectively having p and q as probability distribution vectors, then X is said to be less than Y in the strong stochastic sense, that is, $X \preceq_{st} Y$ iff $\sum_{j=k}^{n} p_j \leq \sum_{j=k}^{n} q_j$ for $k = 1, 2, \ldots, n$.*

Bounds on the distribution imply bounds on performance measures that are non-decreasing functions of the state indices. Observe that performance measures such as average population size, tail probabilities and so on are non-decreasing functions. In our context, the reward that we wish to bound (i.e., $1 - A$) is a non-decreasing function once the state space has been correctly ordered. Let us now illustrate definition 1 by an example:

Example 1. Let $\alpha = (0.1, 0.3, 0.4, 0.2)$ and $\beta = (0.1, 0.1, 0.5, 0.3)$. It follows then that $\alpha \preceq_{st} \beta$ since:

$$\begin{bmatrix} 0.2 & \leq 0.3 \\ 0.2 + 0.4 & \leq 0.3 + 0.5 \\ 0.2 + 0.4 + 0.3 & \leq 0.3 + 0.5 + 0.1 \end{bmatrix}$$

It has been known for some time that monotonicity [6] and comparability of transition probability matrices yield sufficient conditions for the stochastic comparison of Markov chains and their transient and steady-state distributions. Furthermore, st-monotonicity and st-comparability of matrices may be completely characterized by linear algebraic constraints [6].

Definition 2 (St-comparison of transition matrices). *Let P and R be two transition matrices. P is st-smaller than R if and only if $\sum_{k=j}^{n} P_{i,k} \leq \sum_{k=j}^{n} R_{i,k}$ for all k and i between 1 and n.*

Definition 3 (St-monotonicity of transition matrix). *Let P be a transition matrix. P is st-monotone if and only if $\sum_{k=j}^{n} P_{i-1,k} \leq \sum_{k=j}^{n} P_{i,k}$, for all k between 1 and n and for all i between 2 and n.*

We present now the relevant theorem for the stochastic comparison of Markov chains [14]. The statement below assumes that P, the original matrix we want to bound, is not monotone and that we want to obtain an upper bound.

Theorem 1. *Let $X(t)$ and $Y(t)$ be two DTMC and P and R be their respective stochastic matrices. If*

- *$X(0) \preceq_{st} Y(0)$,*
- *R is st-monotone,*
- *$P \preceq_{st} R$,*

then $X(t) \preceq_{st} Y(t)$, for all $t > 0$. If X and Y have steady-state distributions π_X and π_Y, then $\pi_X \preceq_{st} \pi_Y$.

Using this theorem and assuming that we want to compute an upper bound for P, which is the transition matrix of the problem we have to solve, we must find R such that:

$$\begin{cases} \sum_{k=j}^{n} P_{i,k} \leq \sum_{k=j}^{n} R_{i,k}, & \forall\, i, j, \\ \sum_{k=j}^{n} R_{i,k} \leq \sum_{k=j}^{n} R_{i+1,k}, & \forall\, i < n, j. \end{cases} \tag{1}$$

The first set of inequalities states that P is stochastically smaller than R while the second set shows that R is st-monotone. But these two sets of constraints do not help for the numerical resolution for transient or steady-state expected rewards. Thus we also impose additional restrictions on the structure of R in order to facilitate the computation of the bounds. Specifically, we shall insist here that the matrix R be ordinary lumpable.

Definition 4. (Ordinary lumpability) *Let P be the transition probability matrix of an irreducible finite DTMC and let C_k, $k = 1, 2, \ldots, M$ be a partition defined on the states of this Markov chain. Thus, each state of the Markov chain belongs to one and only one of the so-called* macro-states C_k. *The chain is said to be* ordinary lumpable *with respect to the partition C_k if and only if, for all states e and f belonging to the same arbitrary macro state C_k, we have*

$$\sum_{j \in C_i} p_{e,j} = \sum_{j \in C_i} p_{f,j}, \quad \text{for all macro states} \quad C_i, \quad i = 1, 2, \ldots, M. \tag{2}$$

Fourneau et al.[8] have shown that ordinary lumpability constraints are consistent with the relations specified by equation (1). Furthermore, they have designed and implemented an algorithm, called LIMSUB, which constructs a matrix R that possesses all these properties. The lumped matrix is much much smaller than the original matrix. This lumped matrix is readily solved and the bounds obtained from it may be applied to the original Markov chain.

We will now show how Theorem 1 and a slightly modified version of this algorithm establish a common methodology for computing both transient and steady-state bounds.

3 Bounds for Reliability: extending the theory

Our new LMSUB algorithm is based on LIMSUB [8] algorithm. Unfolding relations (1), and satisfing relations (2) for the bounding matrix and a given partition, we obtain a lumpable, st-monotone upper bound. The proof of this new algorithm is almost identical to the proof of LIMSUB algorithm so we refer the reader to [8]. LMSUB algorithm, hovever, does not care about irreducibility and can be, therefore, used to compute bounds of reducible matrices.

We will illustrate this algorithm on the following example. Assume that the chain has 5 states and the state-space is partitioned into two macro-states: $(1, 2)$ and $(3, 4, 5)$. Clearly, relations (1) allow that we compute the lumped matrix column by column. And we must perform some additional computations at the boundaries of the blocks to insure that the matrix is lumpable. In equations (1) we replace inequalities by equalities during the first step. The relations are unrolled and the equalities are arranged in increasing order for i and in decreasing order for j. During the second step, we must modify the first column of the block to insure that each block has a constant row sum. The matrices below show the initial matrix (on the left), then the matrix after the computation of three columns using step 1, the modification of these elements due to the second step, and finally the lumped matrix (on the right). The values modified during the second step are boxed.

$$\begin{bmatrix} 0.2 & 0.2 & 0.1 & 0.3 & 0.2 \\ 0.1 & 0.2 & 0.1 & 0.5 & 0.1 \\ 0.0 & 0.3 & 0.5 & 0.1 & 0.1 \\ 0.1 & 0.2 & 0.4 & 0.3 & 0.0 \\ 0.0 & 0.1 & 0.0 & 0.9 & 0.0 \end{bmatrix} \quad \left[\begin{array}{cc|ccc} & & 0.1 & 0.3 & 0.2 \\ & & 0.1 & 0.4 & 0.2 \\ \hline & & 0.1 & 0.4 & 0.2 \\ & & 0.1 & 0.4 & 0.2 \\ & & 0.0 & 0.7 & 0.2 \end{array}\right] \quad \left[\begin{array}{cc|ccc} & & \boxed{0.2} & 0.3 & 0.2 \\ & & 0.1 & 0.4 & 0.2 \\ \hline & & \boxed{0.3} & 0.4 & 0.2 \\ & & \boxed{0.3} & 0.4 & 0.2 \\ & & 0.0 & 0.7 & 0.2 \end{array}\right] \quad \begin{bmatrix} 0.3 & 0.7 \\ 0.1 & 0.9 \end{bmatrix}$$

In Fourneau et al.[8], only the comparison of steady-state distributions was considered. However, theorem 1 states that the sample-paths are ordered. Thus the comparison of distributions is also true for transient distributions and rewards. And it is even not necessary that the chains are irreducible. It is possible to use this theorem to compare probability of reaching an absorbing state. This is particularly useful when we want to bound the reliability $R(t)$ because we only consider the restriction of the initial matrix to the UP states and one absorbing DOWN state.

We know that "st"-bounds are associated with non-decreasing rewards. Then, if $X(t) \preceq_{st} Y(t)$ at time t and $r(i)$ is a non-decreasing reward function, it follows that

$$\sum_i r(i)Prob(X(t) = i) \leq \sum_i r(i)Prob(Y(t) = i).$$

Now suppose that we use any algorithm which builds a lumpable upper bound. Let C_p be an arbitrary macro-state of the partition we have used to build the bound. Let us now design a new reward function $s(p)$ as the maximum of $r(i)$ for states i in C_p. Clearly, we have two important properties:

Property 1 *$s(p)$ is non-decreasing because the states are initially ordered according to the macro-state and $r(i)$ is non-decreasing.*

Property 2 *At each time step t, the probability of being in macro state C_p multiplied by the reward $s(p)$ is greater than the sum of the individual rewards multiplied by the probabilities of all the states in macro state C_p:*

$$s(p)Prob(Y(t) \in C_p) \geq \sum_{i \in C_p} r(i)Prob(Y(t) = i).$$

As the stochastic matrix associated with Y is lumpable, the left hand-side of the former inequality can be computed using the *lumped* chain Z. Combining both inequalities we get, for all t,

$$\sum_i r(i)Prob(X(t) = i) \leq \sum_p s(p)Prob(Z(t) = p).$$

Putting everything together we obtain the following more general result concerning our algorithms.

Theorem 2. *Let X be a finite aperiodic DTMC and let $r()$ be non-decreasing rewards defined on the states of X. Consider an arbitrary partition of the state space such that states which belong to the same macro-state are contiguous.*

Let Y be the finite DTMC obtained by LMSUB. Y is lumpable and let Z be the lumped version of Y. Assume that $X_0 \preceq_{st} Y_0$. We define the rewards $s()$ at the macro-state level as the maximal reward for the individual states. Then:

- *For any instant t, the expected reward at time t $E_X(r)(t)$ is upper bounded by the expected reward $E_Z(s)(t)$.*

– *The steady-state reward $E_X(r)$ is upper bounded by $E_Z(s)$.*

And both computations $E_Z(r)(t)$ and $E_Z(r)$ require working on matrix Z which is much smaller.

Let us now turn back to the reliability and point availability problem. Clearly, we have two values for the reward: 0 and 1. So we suggest the following for the partition and the corollary it clearly implies:

Rule 1 *Do not group in the same macro-state* UP *and* DOWN *states.*

Corollary 1 *Using this rule, it is not even necessary to compute the maximum and we bound directly the point availability, the reliability and steady-state availability of X by the same values computed on lumped matrix Z.*

3.1 Avoiding the generation of the whole state space

The fundamental requirement is the existence of the transition matrix on the Markov chain on disk. But for some problem of reliability of multicomponent systems, this is even not possible to generate and store the state space and the transition matrix. For instance, the system studied by Muntz [11] and Carrasco [1] has more than 9.0 10^{10} states and 10^{12} transitions. Thus the matrix stored in sparse format represents more than one terabyte. Clearly, alternative description based on tensor product [12] or MTMDD [2] may be useful for the transition matrix. But in our problem even the state space is too large.

So, instead of generating the initial matrix using the visit of reachable states from an initial one with a BFS (Breadth First Search) algorithm, we design a new algorithm (called LL for Lumpable, and Larger) to build a matrix which is larger in the stochastic sense and which is lumpable. Of course, we only build and store the lumped matrix. We obtain a transition probability matrix as we also perform the uniformization process during the generation. It is worthy to remark that this matrix is not monotone in general. This matrix will be the input of LMSUB algorithm in the next step. So we perform two aggregations of the chain before the analysis.

A careful inspection of this state space shows that most of the states are DOWN states. So we use the following rules to design the first step macro-states:

Rule 2 *Do not aggregate the* UP *states during the first step.*

Rule 3 *During the first step aggregate the* DOWN *states which have the same total number of faults.*

Now we have to find the transition probabilities within this new chain. Here, we assume that the description of the model is based on events: an event has a rate and when we apply an event to a state, we obtain the resulting state. The rate does not depend on the states. These assumptions are used to explain how we group transitions. They are not necessary and the same work can be done with other formalisms as well. For the sake of concision, it is not possible to give a proved version of the algorithm here. Algorithm LL is based on the following ideas to obtain a lumpable larger bound:

- Do not change the transition probabilities between simple states.
- The transition from a simple state x to an aggregated state C_p is the sum of the transition probabilities from x to y, for all y in C_p.
- For transitions leaving an aggregated state C_p to an aggregated state C_q (if C_q is a single state, just modify step 4).
  1. label all transitions with the events,
  2. group the transitions and the events according to the number of failures (for instance, a "+1" transition models a new fault),
  3. if an event is associated to two (or more) numbers of failures, then modify the transitions as follows: all the transitions labeled with this event must now join the largest state reached by this event from a state in macro-state C_p.
     For instance, if event u is associated to one or two new faults, then modify the transitions such that now event u is always two new faults.
  4. Then do the summation for all the states in C_q.

Finally we perform the uniformization. Clearly, this algorithm is problem dependent. However, from this specification, we can clearly state that the matrix is larger in the stochastic sense (we move transitions to upper states) and lumpable.

Finally, the total comparison process does not depend of the algorithm used to obtain a lumpable stochastically larger matrix. And clearly the bound obtained by LMSUB or LIMSUB of the matrix we obtain is also a bound of the original matrix we are not able to store.

4 Improving the algorithms

Even if LIMSUB algorithm described in [8] and our new algorithm are closely related, there are several points concerning the implementation which differ considerably. In this section we present the main modifications that speed up the algorithm, especially in case of a transition matrix with relatively few non-zero elements per state, compared to the size of the state space. It also allows the computation of a bound of a reducible transition matrix which is necessary in our approach to bound the reliability of repairable systems.

4.1 LMSUB, LIMSUB and the irreducibility issue

In [8] only the irreducible transition matrices with some additional properties (see [8], Theorem 3) have been considered. To ensure that the irreducibility property is maintained by LIMSUB algorithm the authors in [8] avoid deleting transitions and, if necessary, add small sub-diagonal transitions.

When computing the bounds for transient distributions, we might want to compute the bounds even for the reducible matrices. In order to obtain the reliability bounds using our approach, for instance, we need to compute an st-monotone upper bound of a matrix having one absorbing state (corresponding to the DOWN states of the model). In our implementation of this algorithm we leave the choice to the user if the bound to be computed needs to be irreducible or not.

4.2 Avoiding Computations

The algorithm computes the bounding matrix column per column beginning with the last column. It is clear that it is necessary to store only one column of the matrix P at a time. Only after the first step (Algorithm 1) we know how to modify the first column of the block to obtain a constant row sum. Furthermore due to st-monotonicity, we know that the maximal row sum is obtained with the last row of the block.

```
        for bloc=M to 1 do
step1:      for column=last(block) to first(block) do
                computeCol(column);
            end
step2:      endBlock();
        end
```

Algorithm 1: LIMSUB algorithm [8]

We keep the first step as described in [8]. For the second step, however, there is no need to recompute the first column of a block as all the information needed to compute the next column is only the vector of partial sums $ps_Q^{(j)} = (\sum_{k=j}^{n} q_{i,k})_{i=1}^{n}$ of the lumpable bound Q for the current column j. As the lumpability imposes that this sum is constant for all the rows of a block and, due to the st-monotonicity constraints, we know that the maximal value is obtained for the last row of a block, we need only to store one single value per block. This value is already known at the end of step 1 of the algorithm, so the second step can be completely avoided.

4.3 Sparse Matrix Implementation

The repairable system models have often a huge state space but relatively few transitions per state, which makes interesting to use the sparse representation of transition matrices. Our implementation of LMSUB algorithm exploits this representation and uses the adapted data structures to reduce further the computations. For instance, for the vector $ps_Q^{(j)}$ of partial sums for the current column j of the lumpable bound Q, we store only the index and the value for the rows this sum strictly increases (the elements of $ps_Q^{(j)}$ are increasing because of the st-monotonicity constraints), i.e we store $(i, ps_Q^{(j)})$ if and only if $\sum_{k=j}^{n} q_{i,k} - \sum_{k=j}^{n} q_{i-1,k} > 0$ (with $\sum_{k=j}^{n} q_{-1,k} = 0$). Notice that it is necessary to use only one such structure (a list for example) during the whole computation process as the elements of $ps_Q^{(j)}$ are computed in increasing order, so the old elements, corresponding to the column $j-1$ with indices smaller than the current position, are no longer needed. This allows us to compute only the elements of $ps_Q^{(j)}$ whose value is different from $ps_Q^{(j+1)}$.

Furthermore, we know that those changes are due to the non-zero elements in the column j of the original matrix P (st-comparison between P and Q). Between the two non-zero elements of P at the positions denoted by i_1 et i_2, it is only necessary to update the list containing the information on $ps_Q^{(j)}$ vector, i.e. to erase some elements if they are smaller or equal to the last computed element (at position i_1).

We illustrate this on the example below. On the left is the initial matrix and on the right the bound obtained by LMSUB algorithm. In the table in the middle the first column represents the current column, the second the number of computations performed for that column and the last one the sparse-vector $ps_Q^{(j)}$ throughout the computation process. One can notice that the number of computations performed is sometimes even smaller (column 2) than the number of non-zero elements in the corresponding column of the initial matrix. This is the consequence of the fact that, once the partial sum of value 1 is encountered, the computation of the current column is finished. Note that the "lump" steps do not require computation following the previous remarks.

$$\left[\begin{array}{cc|ccc|cc}
0.3 & 0.6 & 0 & 0 & 0.1 & 0 & 0 \\
0.4 & 0 & 0.5 & 0 & 0 & 0.1 & 0 \\
\hline
0.1 & 0 & 0.3 & 0.4 & 0 & 0 & 0.2 \\
0 & 0.7 & 0 & 0 & 0.3 & 0 & 0 \\
0.1 & 0 & 0.5 & 0 & 0 & 0.4 & 0 \\
\hline
0 & 0 & 0 & 0.8 & 0 & 0.2 & 0 \\
0 & 0.5 & 0 & 0 & 0.1 & 0.3 & 0.1
\end{array}\right]$$

col.	comp.	sparse vector $ps_Q^{(j)}$
7	2	$\{(3, 0.2)\}$
6	4	$\{(2, 0.1), (3, 0.2), (5, 0.4)\}$
lump:	0	$\{(1, 0.1), (3, 0.4)\}$
5	3	$\{(1, 0.1), (3, 0.4), (7, 0.5)\}$
4	2	$\{(1, 0.1), (3, 0.6), (6, 1)\}$
3	3	$\{(1, 0.1), (2, 0.6), (3, 0.9), (6, 1)\}$
lump:	0	$\{(1, 0.6), (3, 0.9), (6, 1)\}$
2	2	$\{(1, 0.7), (3, 0.9), (4, 1)\}$
1	0	$\{(1, 1)\}$
lump:	0	$\{(1, 1)\}$

$$\begin{bmatrix} 0.4 & 0.5 & 0.1 \\ 0.1 & 0.5 & 0.4 \\ 0 & 0.6 & 0.4 \end{bmatrix}$$

5 Numerical Results

In this section we give some numerical results for two examples of repairable systems [1, 11]. The first example is rather small and it is presented here in order to illustrate the quality of our bounds as it is possible to solve the transient and steady-state distributions of the original system. We use LMSUB and LIMSUB but LL us useless. The second example has more than 9×10^{10} states with the number of transitions of order of 10^{12}. So we are not even able to generate the whole transition rate matrix. Yet it is still possible to provide the bounds both for transient and steady-state rewards using our approach.

In the first example we have a system composed of: a front-end (FE), a database (DB), and two processing subsystems having each a switch (S), a memory (M) and two processors (P). The system is operational if it is possible to

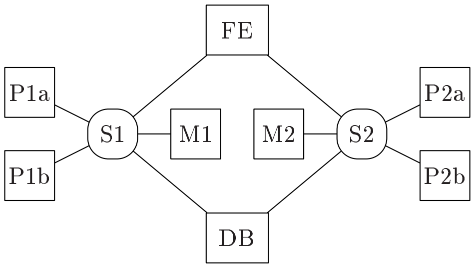

Fig. 2. First example.

access the database i.e. if front-end, database and at least one processing subsystem are operational. A processing subsystem is operational if the corresponding switch, memory and at least one processor are operational. The failures and reparations of components are modeled by exponential distributions. The failure rates are $1/120h^{-1}$ for processors and $1/2400h^{-1}$ for other components and repair rates are $1h^{-1}$ for all the components. A processor failure contaminates the database with probability 0.01. Once the system is down the components do not fail. The components are repaired by a single repairman with the priority given to front-end and database, followed by switches and memories, and then the processors. Within the same priority group the components are chosen at random. We consider the preemptive policy.

The original model has 142 states from which 32 are operational. The maximal number of failed components is 7. Notice that for our approach the order of states is important as all the UP states must precede the DOWN states. Furthermore, we take into account the fact that the LMSUB algorithm yields better bounds for the initial matrix which is almost st-monotone. The choice of partition is also very important as the lumpability step of the algorithm performs much better if the states within the same block have similar properties.

Within the same class of states (UP or DOWN) we ordered the states according to the number of failed components. We chose the following partition : all the states of the same class with the same number of failed components form a block, except the states with only one failure which are left as single-state blocks. This gives 17 blocks, 9 of them composed of UP states.

Solving the original model we obtained steady-state availability $A = 0.998835$. The lower bound obtained by our method is 0.998667. In figure 3 we present the transient bounds for point availability and reliability for this example. We can notice that both results are really close to the exact values.

The second example is presented in figure 1. The system is operational if at least one of processor PA or PB is operational, at least one controller of each type and at least three out of four disks of each of the six clusters are operational. Only one processor of each type is active and only the active processors can fail. A failure of active processor PA is propagated to the active processor PB with probability 0.1. The failures and reparations of components are exponentially distributed. The failure rates are given in table 1. There are two failure modes (soft and hard) which occur with equal probability. When the system is opera-

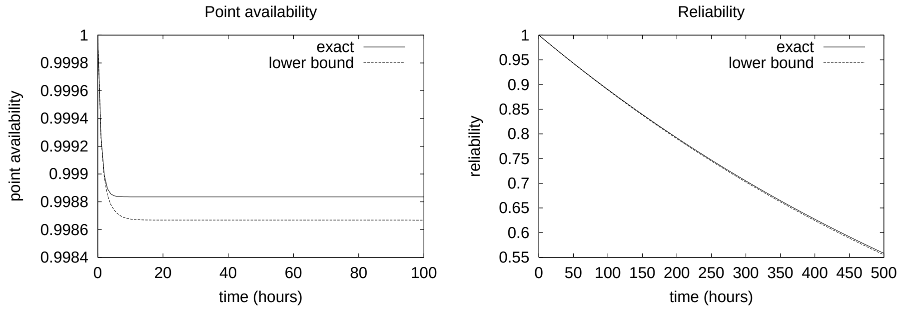

Fig. 3. The exact point availability (left) and reliability (right), and their lower bounds for the first example.

tional the repair rates are $0.1h^{-1}$ in the soft mode and $0.05h^{-1}$ in the hard mode. When the system is down, the rates are 10 times larger as the consequence of the additional precautions to be taken when the system is operational. There is only one repairman who chooses the component to be repaired at random from the set of failed components.

Component	PA, PB and C1	C2	D1	D2	D3	D4	D5	D6
Failure Rate	1/2000	1/4000	1/6000	1/8000	1/10000	1/12000	1/14000	1/16000

Table 1. Failure rates (h^{-1}) for the second example.

The model has only 36 components of 10 different types yet the state-space is of order of 9.0 10^{10}. We used the technique described in section 3.1 to reduce the state space. All the UP states are generated and the DOWN states are aggregated according to the number of failed components. This gives a new model with 1312235 states (1312200 UP and 35 DOWN macro-states) and 25754089 transitions.

First let us consider the ordering of the states and the edge between UP states and DOWN states. We have chosen the UP state with the maximal number (15) of simultaneous failures for the system to still be operational, in which all the failures are hard failures and the only operational processor is PB, to be the last UP ($last_{up}$) state. Let D_k denote the DOWN macro-state with k failures. Then for all $D_k, k \geq 3$ there is a transition (D_k, D_{k-1}) and the only transition from DOWN macro states to UP states is the transition $(D_2, last_{up})$. Also, there are transitions $(D_k, D_{k+1}), \forall k < 36$ and $(D_k, D_{k+2}), \forall k < 35$, so the new transition matrix is irreducible. It is also aperiodic due to the uniformization constant $\lambda > \max_i\{-Q_{i,i}\}$, so we can use the optimized version of LIMSUB algorithm.

In the second step the UP states are reordered increasingly in number of failed components followed by number of hard failures. The UP states are followed by the DOWN states ordered increasingly in number of failed components. The partition contains 172 subsets: all the UP states, except $last_{up}$ state, with the same number of failed components and the same number of hard failures are aggregated forming 136 blocks followed by $last_{up}$ state then by 35 one-state blocks corresponding to the DOWN states, already aggregated in the first step.

The lower bound for steady-state availability of the second system obtained by this method is 0.999132158 (upper bound for unavailability is 0.000867842). The lower bounds for point availability are given in figure 4 (left).

In table 2 we give computational times for all three steps. For the third step we also report the time needed by LIMSUB [8] algorithm (on the same machine). We can notice that on this example our algorithm is approximately twice faster. This is a consequence of the improvement made during the normalization part of the algorithm as well as the better utilization of the matrix sparse structure.

When we bound the reliability, we are only interested in UP states. We are computing the lower bound for reliability with the st-monotone upper bound of the chain. We obtain this bound by aggregating all the DOWN states into one absorbing state. We ordered UP states and regrouped them into blocks according to the number of failed components followed by the number of hard failures. This gives us a partition into 137 blocks: 136 corresponding to UP states and 1 to the absorbing DOWN state. The lower bounds for reliability for the second system are reported in figure 4 (right).

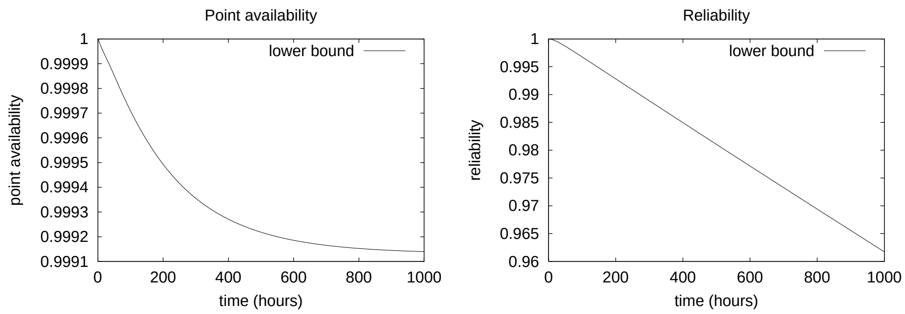

Fig. 4. Lower bounds for point availability and reliability for the second example.

generation	reordering	LMSUB (ird)	LIMSUB [8]
182.0	95.52	56.54	107.3

Table 2. CPU times in seconds on a PC (CPU 3.20GHz, 1 GB of RAM) to compute the point availability for the second example.

6 Conclusion

In this paper we have extended the theory of algorithmic bounds to reliability and availability problems. The theory now include transient and steady-state analysis and the Markov chains may be irreducible or have an absorbing state. We have also improved the algorithms to be more efficient as the chains are very large. We have also suggested that high level formalisms may be used to build lumpable matrices which are larger in the stochastic sense. Of course, only the lumped matrix is generated and stored. A version of this algorithm based on Stochastic Automata Network is currently under development.

References

1. J.A. Carrasco, "Bounding steady-state availability models with group repair and phase type repair distributions", Performance Evaluation, V35 (1999), 193–204.
2. G. Ciardo and A.S. Miner. "Storage alternatives for larger structured state spaces." In R. Marie, B. Plateau, M.Calzarossa, and G. Rubino, editors, Proc. 9th Int. Conf. on Modelling Techniques and Tools for Computer Performance Evaluation, LNCS 1245, pages 44-57, St. Malo, France, June 1997. Springer-Verlag.
3. P.J. Courtois, P. Semal, "Bounds for the positive eigenvectors of nonnegative matrices and their approximations", J. ACM, V31 (4) (1984), 804–825.
4. T. Dayar, J.M. Fourneau, N. Pekergin, "Transforming stochastic matrices for stochastic comparison with the st-order", RAIRO-RO, V37 (2003), 85–97.
5. T. Dayar, N. Pekergin: "Stochastic comparison, reorderings, and nearly completely decomposable Markov chains." In: Proceedings of the International Conference on the Numerical Solution of Markov Chains (NSMC'99), (Ed. Plateau, B. Stewart, W.), Prensas universitarias de Zaragoza. (1999) 228-246.
6. J.M. Fourneau, N. Pekergin. "An algorithmic approach to stochastic bounds", LNCS 2459, Performance evaluation of complex systems: Techniques and Tools, pp 64–88, 2002.
7. J.M. Fourneau, M. Lecoz, N. Pekergin and F. Quessette, "An open tool to compute stochastic bounds on steady-state distributiuon and rewards", IEEE Mascots 2003, Orlando, USA.
8. J.M. Fourneau, M. Lecoz, F. Quessette, "Algorithms for an irreducible and lumpable strong stochastic bound", Numerical Solution of Markov Chains, 2003, USA.
9. J.C.S. Lui, R. Muntz, "Computing bounds on steady state availability of repairable computer systems", J. ACM, V41 (4) (1994), 676–707.
10. S. Mahevas, G. Rubino, "Bounding asymptotic dependability and performance measures", Second IEEE International Performance and Dependability Symposium, USA, 1996, 176–186.
11. R. Muntz, E. de Souza e Silva, A. Goyal, "Bounding availability of repairable computer systems", IEEE Trans. on Computers, V38 (12) (1989), 1714–1723.
12. B. Plateau, "On the stochastic structure of parallelism and synchronization models for distributed algorithms", Proc. of the SIGMETRICS Conference, Texas, 1985, 147–154.
13. W.J. Stewart, "Introduction to the Numerical Solution of Markov Chains", Princeton University Press, 1994.
14. D. Stoyan, "Comparison Methods for Queues and Other Stochastic Models", Wiley, 1983.