An open tool to compute stochastic bounds on steady-state distributions and rewards

Jean-Michel Fourneau, Mathieu Le Coz, Nihal Pekergin and Franck Quessette

PRiSM Laboratory, Versailles University
45, avenue des États-Unis, 78000 Versailles, FRANCE
Email: {jmf,mlc,nih,qst}@prism.uvsq.fr

Abstract

We present X-Bounds, a new tool to implement a methodology based on stochastic ordering, algorithmic derivation of simpler Markov chains and numerical analysis of these chains. The performance indices defined by reward functions are stochastically bounded by reward functions computed on much simpler or smaller Markov chains obtained after aggregation or simplification. This leads to an important reduction on numerical complexity. Typically, chains are ten times smaller and the accuracy may be good enough.

1. Introduction and Motivation

Since Plateau's pioneering work on tensorial representation of Markov chains [14], we know how to design and store huge states spaces and transition sets. Many high level formalisms such as Petri nets and Stochastic Process Algebra are able now to store the transition matrix P of a chain in a tensor form [7, 12]:

$$P = \sum_i \bigotimes_j M_i^j$$

However, this method does not help enough during the numerical resolution of the steady-state. More precisely, we are interested in performance indices or rewards R defined as summation of elementary reward functions $r(i)$ on the steady-state distribution π (i.e. $R = \sum_i r(i)\pi(i)$) and we do not try to compute transient measures. Important performance indices such as mean population size, average waiting time or loss probabilities may be computed that way. Thus the numerical computation is mainly the computation of the steady-state distribution and the summation of the elementary rewards $r(i)\pi(i)$ to obtain R. π is defined by $\pi = \pi P$. This first step is in general the most difficult because of the memory space and time requirements (see Stewart's book [15] for an overview of usual numerical techniques for Markov chains).

We advocate algorithmic bounds as an interesting alternative to computation of the exact steady-state distribution when it is sufficient to satisfy the requirements of the Quality of Service (QoS) we expect. Here, we present a tool and some algorithms to check several algorithmic bounds based on stochastic ordering of Markov chains. The approach is quite recent as stochastic bounds are usually obtained by sample-path proofs rather than by algorithmic derivation (see [13] for an example on Fair Queueing). The algorithmic aspects of stochastic comparison of Markov chains have been surveyed recently [9] and the tool we present implements many of the algorithms on this new topic. For some examples, we have obtained very good results, the state space is reduced by several order of magnitude (from several millions to several tens of thousands) and the bounds are still accurate. The availability of this tool will help to compare the accuracy of bounding techniques. The tool is open to help the modification of algorithms or heuristics and the source will be available.

The paper is organized as follows: in section 2, we briefly introduce the properties we need and the basic algorithms and results already obtained. Then section 3 is devoted to the presentation of the tool, the states and transition generation and the reordering. Finally in section 4, we present the bounding methods already proved [2, 8, 9] and an example.

2. A Brief Overview of Stochastic Bounds and Algorithms

We restrict ourselves to Discrete Time Markov Chains (DTMC) with finite state space $E = \{1, 2, \ldots, n\}$ but continuous-time models can be considered after uniformization. n and m will be respectively the number of states and the number of non-zero transitions. $P_{i,*}$ will denote row i of matrix P.

Stoyan [16] defined the strong stochastic ordering ("st"-ordering for short) by the set of non-decreasing functions.

Bounds on the distribution imply bounds on these performance indices as well. Important performance indices such as average population, loss rates or tail probabilities are non decreasing functions. The second part of the definition 1 for discrete random variables is much more convenient for an algebraic formulation and an algorithmic setting.

Definition 1 *Let X and Y be random variables taking values on a totally ordered space. Then X is said to be less than Y in the strong stochastic sense, that is, $X <_{st} Y$ iff $E[f(X)] \leq E[f(Y)]$ for all non decreasing functions f whenever the expectations exist.*

If X and Y take values on the finite state space $\{1, 2, \ldots, n\}$ with p and q as probability distribution vectors, then X is said to be less than Y in the strong stochastic sense, that is, $X <_{st} Y$ iff $\sum_{j=k}^{n} p_j \leq \sum_{j=k}^{n} q_j$ for $k = 1, 2, \ldots, n$.

Example 1 *Let $\alpha = (0.1, 0.3, 0.4, 0.2)$ and $\beta = (0.1, 0.1, 0.5, 0.3)$ two probability distribution vectors. We have $\alpha <_{st} \beta$ since:*

$$\left\{ \begin{array}{rcl} 0.2 & \leq & 0.3 \\ 0.4 + 0.2 & \leq & 0.5 + 0.3 \\ 0.3 + 0.4 + 0.2 & \leq & 0.1 + 0.5 + 0.3 \end{array} \right.$$

It is known for a long time that monotonicity [9] and comparability of the transition probability matrices yield sufficient conditions for the stochastic comparison of Markov chains and their steady-state distributions. St-monotonicity and st-comparability of matrices are completely characterized by linear algebraic constraints; this fundamental result is the key of our algorithms. Assuming that P is not st-monotone, we must find Q such that:

$$\left\{ \begin{array}{rcll} \sum_{k=j}^{n} P_{i,k} & \leq & \sum_{k=j}^{n} Q_{i,k} & \forall\, i, j \\ \sum_{k=j}^{n} Q_{i,k} & \leq & \sum_{k=j}^{n} Q_{i+1,k} & \forall\, i, j \end{array} \right. \quad (1)$$

The first equation states that P is stochastically lesser than matrix Q, the second gives the condition for Q to be a st-monotone matrix.

We will have $\pi_P <_{st} \pi_Q$ if the steady-state distributions exist (see [8, 9] for the theoretical aspects). Vincent's algorithm [1] is the simplest solution: it replaces the inequalities by equalities and computes Q with i in increasing order and j in decreasing order. However, this algorithm has two important drawbacks: the result matrix may be reducible even if matrix P is not. And the bounding matrix is in general as difficult to solve as the original one.

We have proved several algorithms to cope with these problems. First, Algorithm *IMSUB* fixes the irreducibility problem (see [8] for a proof). It avoids to delete transitions and it adds the sub-diagonal elements. Then, we use the two sets of constraints of system (1) and add some structural properties of the chain to simplify the resolution. For instance, Algorithm *UHMSUB* (see figure 1) provides an upper bounding matrix which is Upper-Hessenberg (i.e. the low triangle except the main sub-diagonal is zero). Therefore the resolution by direct elimination is quite simple.

Example 2 *Consider the following matrices:*

$$P = \frac{1}{10} \cdot \begin{bmatrix} 5 & 2 & 1 & 2 & 0 \\ 1 & 7 & 1 & 0 & 1 \\ 2 & 1 & 5 & 2 & 0 \\ 1 & 0 & 1 & 7 & 1 \\ 0 & 2 & 2 & 1 & 5 \end{bmatrix} \quad Q = \frac{1}{10} \cdot \begin{bmatrix} 5 & 2 & 1 & 2 & 0 \\ 1 & 6 & 1 & 1 & 1 \\ 0 & 3 & 5 & 1 & 1 \\ 0 & 0 & 2 & 7 & 1 \\ 0 & 0 & 0 & 5 & 5 \end{bmatrix}$$

$$R = \frac{1}{10} \cdot \begin{bmatrix} 5 & 2 & 1 & 2 & 0 \\ 1 & 6 & 1 & 1 & 1 \\ 2 & 0 & 6 & 2 & 0 \\ 0 & 0 & 2 & 7 & 1 \\ 0 & 0 & 0 & 5 & 5 \end{bmatrix} \quad S = \frac{1}{10} \cdot \begin{bmatrix} 5 & 2 & 1 & 2 & 0 \\ 1 & 6 & 1 & 1 & 1 \\ 1 & 2 & 5 & 1 & 1 \\ 0 & 0 & 0 & 7 & 3 \\ 0 & 0 & 0 & 5 & 5 \end{bmatrix}$$

The following properties are easy to check:

- *$P <_{st} Q$ and Q is st-monotone and Q is Upper-Hessenberg*
- *$P <_{st} R$ but R is not st-monotone (see the second and the third row of R).*
- *$P <_{st} S$ and S is st-monotone but S is reducible.*

All the algorithms we have integrated into X-Bounds are structure-oriented. They do not assume any particular property or structure for the initial stochastic matrix but the bounds have a structure which helps for the numerical resolution. Let us review the Upper-Hessenberg case.

In the algorithm description (see figure 1), for the sake of simplicity, we use a full matrix representation for P and Q. Of course, the programs use sparse matrix representations. Note that due to the ordering of the indices, the summations are already computed and stored when we need them. We let them appear as summations to show the relations with the set of inequalities (1).

The resolution by recursion for these matrices requires $O(m)$ operations [15]. The bounding algorithm is slightly different of Algorithm *IMSUB* [8]. The last two instructions create the Upper-Hessenberg structure.

We have also developed new techniques to improve the accuracy of the bounds on the steady state π which are based on some pre-processing of P [5] or polynomials of P [4]. These transformations have no effect on the steady-state distribution but they have a large influence on the bounding algorithms. Thus, X-Bounds allows to build the original matrix P or some simple polynomials of P.

3. The Tool

The program interface is divided in five columns (see figure 2), each one representing a step of the resolution.

The first step is to obtain the states and transitions of the chain from some specifications. Then the chain must

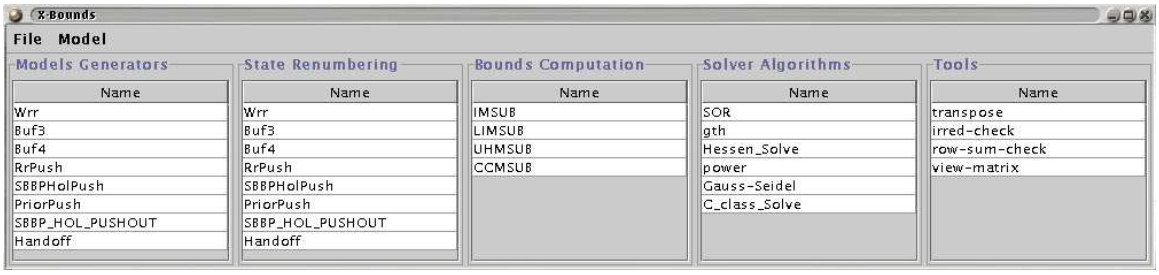

Figure 2. The X-Bounds Interface.

$q_{1,n} = p_{1,n}$;
For $i = 1, 2, \ldots, n$ **Do**
  $q_{1,i} = p_{1,i}$
  $q_{i+1,n} = \max(q_{i,n}, p_{i+1,n})$;
End For
For $i = 2, 3, \ldots, n$ **Do**
  For $l = n-1, n-2, \ldots, i$ **Do**
    $q_{i,l} = \max(\sum_{j=l}^{n} q_{i-1,j}, \sum_{j=l}^{n} p_{i,l}) - \sum_{j=l+1}^{n} q_{i,j}$;
    If $(q_{i,l} = 0)$ and $(p_{i,l} > 0)$ and $(\sum_{j=l+1}^{n} q_{i,j} < 1)$ **Then**
      $q_{i,l} = \epsilon \times (1 - \sum_{j=l+1}^{n} q_{i,j})$
    End If
  End For
  $q_{i,i-1} = 1 - \sum_{j=i}^{n} q_{i,j}$
  For $l = i-2, i-3, \ldots, 1$ **Do**
    $q_{i,l} = 0$
  End For
End For

Figure 1. Algorithm UHMSUB: Construction of an irreducible Upper-Hessenberg st-monotone upper bound.

be processed to satisfy the requirements of the bounding algorithms which have been chosen. Typically, the transition matrix is obtained and stored row by row while most of the algorithms use ad-hoc storage (for instance, *LIMSUB* needs a column by column storage, beginning with the rightmost one). Finally the chain is solved and the rewards are computed.

Generation of reachable states: The first step *Models Generators* generates the matrix files from user defined C-functions.

We proceed by a Breadth-First Search from a chosen initial state (see [10] for such a method for states generated by a Petri net). Here the transitions are described by an evolution equation based on states and events. The states are described by a multidimensional vector, therefore they are included into a Cartesian product. Each transition is associated to an event. An event is described by a probability which may be state-dependent and by the transitions it triggers for all the states. So, the user has to modify four functions, written in C, to specify: the Cartesian product including the states, the initial state to perform the visit, the probability of a transition, and the evolution equation which describes all the transitions. The functions are gathered into a file "fun.c". Once these functions have been provided, the program is compiled to obtain a new stochastic matrix generation tool which is stored in the model directory. It is also possible to define (using a set of constants in a file "const.h") a polynomial Φ to compute $\Phi(P)$ during the generation rather than P.

Figure 3 depicts the generation algorithm. The sets S and V and the set operations $+$, $-$ and $AddTransition$ must me carefully implemented. The sets are represented by queues, and the transitions are linked in an ordered list as we assume that the matrix will be sparse. Another implementation based on hash table will also be considered in the near future.

$S = \{Initial\ State\}$;
$V = \emptyset$;
While $S \neq \emptyset$ **Do**
  Take s in S ;
  $S = S - \{s\}$;
  Initialize the list of Transitions ;
  For all event e **Do**
    p = Probability(e,s) ;
    x = Evolution(s,e) ;
    $S = S + \{x\}$;
    AddTransition(x,p) ;
  End For
  Write List of Transitions ;
End While

Figure 3. Algorithm GENERATION: to build the transition matrix.

Reordering the states: The second step *State Renumbering* generates an additional file from user defined C-functions and the output files of the first step. The files generated contain the matrix reordered and the partition of the states.

One of the major assumptions of most algorithms is the ordering of the states. First, the rewards must be a non decreasing function of the state-index. Furthermore, some algorithms based on a partition of states assume that states which belong to the same set have contiguous indices. The accuracy of the bound is directly impacted by the ordering [6] due to the filling of the bounding matrix. Thus we

have to reorder the states of the matrix. Again, the ordering is defined by the user in a C function. This function computes a new number for a state described by the components in the Cartesian product. The numbers may be not distinct to allow the description of sets of states with the same value. The ordering is based on the natural ordering on these values. The compatibility with the rewards is not checked.

Bounding and Solving the Chain: The third step *Bounds Computation* computes a stochastic bound and generates new files describing the bounding matrix. The fourth step *Solver Algorithms* implements classical solvers and outputs the stationary distribution.

Several algorithms have been implemented: *IMSUB*, *UHMSUB* (with Upper-Hessenberg structure), *LIMSUB* (which builds a lumpable chain [8]), *CCMSUB* (a C-Class matrix structure studied by some of us [3] with a closed form steady-state solution which can be computed in $O(n)$ operations).

Remember that the bounds are based on simpler or smaller chains. Smaller chains are obtained because the bounding matrix is lumpable [8]. And they are solved using usual numerical algorithms. The ordinary lumpability insures that we still have a Markov chain after aggregation of macros-states. So the lumped model is easier to solve. GTH, SOR and Gauss-Seidel methods have been implemented. Simpler chains have the same state space but a structure which allows a simple computation of π. Thus they need ad-hoc algorithms. So we have developed solvers for Upper-Hessenberg or C-Class matrices [3, 9].

Basic Tools: Finally, we also add some utility programs: checking the matrix row sum, matrix transposition to use the SOR or Gauss Seidel solvers, map of the matrix to show the non zero elements, irreducibility checking.

The tool is open. It is sufficient to add a new program into the Solvers or Bounds directories to make them available by the interface. Of course the program must be consistent with the file description for the transitions matrices. It is also possible to compute the rewards. The user has to specify the elementary rewards $r(i)$ using a C function.

Since the aim is to deal with chains which are so large that the transition matrix does not fit in memory, storage and complexity issues are critical. The tool is based on a JAVA interface which creates directory, stores data and models files, and the functions relevant to the model. The real operations are written in C, tailored by the user for the models and compiled by the interface to obtain an efficient implementation.

4. An Example

We present all the steps of the analysis of a real example. We give the functions to describe the model, we define the rewards and show some heuristics to justify the bounding strategy. We present the numerical results and the computation times and compare with exact results for small problems.

4.1 The Problem

We analyze a finite buffer and a buffer policy which combines the PushOut mechanism for the space management and a Head Of Line service discipline. We assume that there exist two types of packets with distinct loss rate requirements. The high priority (type H) packets have priority for space but not for service. A low priority packet (type L) which arrives in a full buffer is lost. If the buffer is not full, both types of packets are accepted. The PushOut mechanism specifies that when the buffer is full, an arriving high priority packet pushes out of the buffer a low priority one if there is any in the buffer; otherwise the high priority packet is lost. We assume that packet size and service time are constant, so we have a discrete time model (this is consistent with ATM cells [11]). We also assume that the departure due to service completion always takes place just before the arrivals. The arrivals follow a Batch Process. The buffer size is B, the number of states is $(B+1)(B+2)/2$. We are interested in the expected number of high priority lost packets per slot. Clearly, we have to estimate only few probabilities as the reward is the number of packets which exceed the buffer size.

4.2 The Model and the Matrix

We use the following representation for the state space (T, H) where T is the total number of packets, H is the number of high priority packets. The states are ordered according to a lexicographic non decreasing order. This representation of states is unusual but it has good properties which are shown in the following. To compute transitions and rewards, we define the evolution C function (reported in figure 4) and compile the generation program.

Let us now show the transition matrix. Assume that the buffer size if 5 and the batch arrival size is between 0 and 3. Let $a_{i,j}$ the probability that a batch is made of i packets of type L and j of type H. If we order the states according to the lexicographic ordering, we obtain the matrix P depicted in figure 5. For the sake of readability we have replaced the six last columns by some blocks which follow and with R_3 entries: $\forall i \in 1..5$, $R_3[i,i] = R_3[6,5] = a_{1,0} + a_{2,0} + a_{3,0}$; $\forall i \in 1..4$, $R_3[i,i+1] = a_{0,1} + a_{1,1} + a_{2,1}$; $\forall i \in 1..3$, $R_3[i,i+2] = a_{0,2} + a_{1,2}$; $\forall i \in 1..3$, $R_3[i,i+3] = a_{0,3}$;

```
void Evolution (long *E, long indexevt, long *F, long *R)
{   /* E is the input state, indexevt is the transition number   */
    /* F is the output state and R is the reward                 */
                        /* Components of state variable E and F */
    long S0, S1, a0, a1; /*      0) total number of customer     */
                         /*      1) number of H customers        */
    Int2arrivees(indexevt, &a0, &a1);
    /* a0 : arrivals of H;  a1 : arrivals of L */

    S0 = S1 = 0;
    if (E[0]>0) {S0 = 1; if (E[1]==E[0])) S1=1;} /* who is in service */

    F[0] = E[0] + a0 + a1 - S0;          /* arrivals */
    F[1] = E[1] + a1 - S1;
    R[1] = 0; R[0] = 0;                  /* Pushout, Rewards and loss */
    if (F[0] > BufferSize) { R[0] = F[0]-BufferSize; F[0] = BufferSize; }
    if (F[1] > BufferSize) { R[1] = F[1]-BufferSize; F[1] = BufferSize; }
}
```

Figure 4. `Evolution` **Function.**

$R_3[4,6] = a_{0,2} + a_{1,2} + a_{0,3}$; $R_3[5,6] = R_3[6,6] = a_{0,1} + a_{1,1} + a_{2,1} + a_{0,2} + a_{1,2} + a_{0,3}$ and other entries are zero.

A careful inspection shows that the 15 first rows of the transition matrix are st-monotone. For a model of a larger buffer, this property is still true for the states where the buffer is not full. Let us now turn to the reward function. Let $\mathbb{1}_{\{x\}}$ be an indicator function. The reward is the number of exceeding packet of type H. As the service completion occurs before the arrival, we first check if the packet in service is a type H. Due to the HOL service discipline, this is equal to $\mathbb{1}_{\{H>0\}}\mathbb{1}_{\{T=H\}}$. Then we add the arrivals and we compute the exceeding packets:

$$r(T,H) = \sum_i \sum_j a_{i,j} \max(0, (H - \mathbb{1}_{\{H>0\}}\mathbb{1}_{\{T=H\}} + i + j - B))$$

Note that this function is not increasing. Indeed, assume a maximal batch size of 3, $r(B-1, B-1) = a_{0,3}$ and $r(B,0) = 0$ if $B > 4$. Therefore we define s an upper-bound of r, easily obtained by induction $s(1) = r(1)$ and $s(i) = \max(s(i-1), r(i))$.

4.3 The Bound and the Results

We only present here the application of *LIMSUB* algorithm which is based on a lumpable bound. The key idea to shorten the state space is to avoid the states with large numbers of low priority packets. So, we bound the matrix with an ordinary lumpable matrix using the following macro-states: let F be a parameter, we aggregate in the macro-state $(T, T-F)$ all the states where the second component is smaller than $T - F$. If $H > T - F$ then the lumped state contains only one state (T, H). For instance, consider a buffer of size 5, if we assume that $F = 2$, the states $(4,0)$, $(4,1)$ and $(4,2)$ are aggregated into one macro-state. Let us now present some numerical results. First, we analyze a model with a small buffer to check the accuracy of the bound. The buffer size is 50. The batch size is between 0 and 3. The batch distribution is, in lexicographic order (L,H), $(0.55, 0.1, 0.1, 0.05, 0.05, 0.05, 0.025, 0.025, 0.025, 0.025)$.

F	*Size*	*NZE*	*Reward*
2	150	1,377	$1.8\ 10^{-9}$
5	291	3,633	$1.7\ 10^{-9}$
10	506	5,674	$1.7\ 10^{-9}$
Exact Model	1,326	12,848	$1.2\ 10^{-9}$

Where *Size* is the size of the matrix, *NZE* the number of Non Zero Entries and *Reward* is the number of type H packets lost per slot (remind that here, the algorithm computes an upper bound of the reward). Clearly the bounds are very accurate. We have found several reasons for that property. First the distribution is skewed, almost all the probability mass is concentrated on the states with a small number of packets. Moreover due to the ordering of the states the first part of the initial matrix is already st-monotone. Thus we only have a few modifications in the matrix.

Now we report the execution time to build the transition matrix and the bound. We consider large buffers. We report in table 6 the matrix sizes and Non Zero Entries. The times were measured on an usual PC.

5 Conclusion

Stochastic and numerical bounds are promising techniques for performance evaluation. The major drawback of these methods is the lack of open tools to test algorithms and check their accuracy. We have observed that these methods may be very accurate when the distribution is skewed and when a large part of the initial matrix is already st-monotone. We hope the availability of X-Bounds will help to develop these techniques.

References

[1] O. Abu-Amsha and J.-M. Vincent. An algorithm to bound functionals of markov chains with large state space. In *4th INFORMS Conference on Telecommunications*, Boca Raton, Florida, 1998.

[2] M. Benmammoun, J.-M. Fourenau, N. Pekergin, and A. Troubnikoff. An algorithmic and numerical approach to bound the performance of high speed networks. In *IEEE Mascots*, Fort Worth, USA, 2002.

[3] M. Benmammoun and N. Pekergin. Closed form stochastic bounds on the stationary distribution of markov chains. *Probability in the Engineering and Informational Sciences*, (16):403–426, 2002.

[4] F. Boujdaine, T. Dayar, J.-M. Fourneau, N. Pekergin, S. Saadi, and J. Vincent. A new proof of st-comparison for polynomials of a stochastic matrix. In *Submitted*, 2003.

[5] T. Dayar, J.-M. Fourneau, and N. Pekergin. Transforming stochastic matrices for stochastic comparison with the st-order. *RAIRO-RO, To appear*, 2003.

[6] T. Dayar and N. Pekergin. Stochastic comparison, reorderings, and nearly completely decomposable markov chains. In B. Plateau and W. Stewart, editors, *Proceedings of the International Conference on the Numerical Solution of Markov Chains (NSMC'99)*, pages 228–246, 1999.

$$
P = \left(\begin{array}{cccccccccc|cccccc}
a_{0,0} & a_{1,0} & a_{0,1} & a_{2,0} & a_{1,1} & a_{0,2} & a_{3,0} & a_{2,1} & a_{1,2} & a_{0,3} & & & & & & 0 \\
a_{0,0} & a_{1,0} & a_{0,1} & a_{2,0} & a_{1,1} & a_{0,2} & a_{3,0} & a_{2,1} & a_{1,2} & a_{0,3} & & & & & & 0 \\
a_{0,0} & a_{1,0} & a_{0,1} & a_{2,0} & a_{1,1} & a_{0,2} & a_{3,0} & a_{2,1} & a_{1,2} & a_{0,3} & & & & & & 0 \\
\hline
 & a_{0,0} & & a_{1,0} & a_{0,1} & & a_{2,0} & a_{1,1} & a_{0,2} & & a_{3,0} & a_{2,1} & a_{1,2} & a_{0,3} & & 0 \\
 & & a_{0,0} & & a_{1,0} & a_{0,1} & & a_{2,0} & a_{1,1} & a_{0,2} & & a_{3,0} & a_{2,1} & a_{1,2} & a_{0,3} & 0 \\
 & & a_{0,0} & & a_{1,0} & a_{0,1} & & a_{2,0} & a_{1,1} & a_{0,2} & & a_{3,0} & a_{2,1} & a_{1,2} & a_{0,3} & 0 \\
\hline
 & & & a_{0,0} & & & a_{1,0} & a_{0,1} & & & a_{2,0} & a_{1,1} & a_{0,2} & & & \\
 & & & & a_{0,0} & & & a_{1,0} & a_{0,1} & & & a_{2,0} & a_{1,1} & a_{0,2} & & R_1 \\
 & & & & & a_{0,0} & & & a_{1,0} & a_{0,1} & & & a_{2,0} & a_{1,1} & a_{0,2} & \\
 & & & & & a_{0,0} & & & a_{1,0} & a_{0,1} & & & a_{2,0} & a_{1,1} & a_{0,2} & \\
\hline
 & & & & & & a_{0,0} & & & & a_{1,0} & a_{0,1} & & & & \\
 & & & & & & & a_{0,0} & & & & a_{1,0} & a_{0,1} & & & \\
 & & & & & & & & a_{0,0} & & & & a_{1,0} & a_{0,1} & & R_2 \\
 & & & & & & & & & a_{0,0} & & & & a_{1,0} & a_{0,1} & \\
 & & & & & & & & & a_{0,0} & & & & a_{1,0} & a_{0,1} & \\
\hline
 & & & & & & & & & & a_{0,0} & & & & & \\
 & & & & & & & & & & & a_{0,0} & & & & \\
 & & & & & & & & & & & & a_{0,0} & & & R_3 \\
 & & & & & & & & & & & & & a_{0,0} & & \\
 & & & & & & & & & & & & & & a_{0,0} & \\
 & & & & & & & & & & & & & & a_{0,0} &
\end{array}\right)
$$

$$
R_1 = \begin{pmatrix}
a_{3,0} & a_{2,1} & a_{1,2} & a_{0,3} & \\
 & a_{3,0} & a_{2,1} & a_{1,2} & a_{0,3} \\
 & & a_{3,0} & a_{2,1} & a_{1,2} & a_{0,3} \\
 & & a_{3,0} & a_{2,1} & a_{1,2} & a_{0,3}
\end{pmatrix}
$$

$$
R_2 = \begin{pmatrix}
a_{2,0}{+}a_{3,0} & a_{1,1}{+}a_{2,1} & a_{0,2}{+}a_{1,2} & a_{0,3} & & \\
 & a_{2,0}{+}a_{3,0} & a_{1,1}{+}a_{2,1} & a_{0,2}{+}a_{1,2} & a_{0,3} & \\
 & & a_{2,0}{+}a_{3,0} & a_{1,1}{+}a_{2,1} & a_{0,2}{+}a_{1,2} & a_{0,3} \\
 & & & a_{2,0}{+}a_{3,0} & a_{1,1}{+}a_{2,1} & a_{0,2}{+}a_{1,2}{+}a_{0,3} \\
 & & & a_{2,0}{+}a_{3,0} & a_{1,1}{+}a_{2,1} & a_{0,2}{+}a_{1,2}{+}a_{0,3}
\end{pmatrix}
$$

Figure 5. The example matrix.

B	Size	NZE	Gener. Time (in s)	F	Bound Size	Bound NZE	Reord. Time (in s)	LIMSUB Time (in s)
500	125,751	1,253,498	4.8 s	10	5,456	61,924	6.6 s	14 s
500	125,751	1,253,498	4.8 s	20	10,311	120,079	6.6 s	34 s
1,000	501,501	5,006,998	19 s	10	10,956	124,424	27 s	84 s
1,000	501,501	5,006,998	19 s	20	20,811	242,579	27 s	167 s
2,000	2,003,001	20,013,998	53 s	10	21,956	249,424	105 s	568 s

Figure 6. Results.

[7] S. Donatelli. Superposed generalized stochastic petri nets: definition and efficient solution. In *Proc. 15th Int. Conf. on Application and Theory of Petri Nets*, Zaragoza, Spain, 1994.

[8] J.-M. Fourneau, M. Le Coz, and F. Quessette. Algorithms for an irreducible and lumpable strong stochastic bound. In *Numerical Solution of Markov Chains*, USA, 2003.

[9] J.-M. Fourneau and N. Pekergin. An algorithmic approach to stochastic bounds. In *LNCS 2459, Performance evaluation of complex systems: Techniques and Tools*, pages 64–88, 2002.

[10] B. Haverkort, A. Bell, and H. Bohnenkamp. On the efficient sequential and distributed generation of very large markov chains from stochastic petri nets. In I. Press, editor, *PNPM 99*, pages 12–21, Spain, 1999.

[11] G. Hébuterne and A. Gravey. A space priority queueing mechanism for multiplexing atm channels. *ITC Specialist Seminar, Computer Network and ISDN Systems*, (V20):37–43, 1990.

[12] J. Hillston and L. Kloul. An efficient kronecker representation for pepa models. In *PAPM*, Aachen Germany, 2001.

[13] N. Pekergin. Stochastic delay bounds on fair queueing algorithms. In *Proceedings of INFOCOM'99*, pages 1212–1220, New York, 1999.

[14] B. Plateau. On the stochastic structure of parallelism and synchronization models for distributed algorithms. In *Proc. of the SIGMETRICS Conference*, pages 147–154, Texas, 1985.

[15] W. Stewart. *Introduction to the Numerical Solution of Markov Chains*. Princeton University Press, 1994.

[16] D. Stoyan. *Comparison Methods for Queues and Other Stochastic Models*. Wiley, 1983.