NUMERICAL SOLUTION of MARKOV CHAINS, p. 191–206

Algorithms for an irreducible and lumpable strong stochastic bound

J.M. Fourneau, M. Lecoz, F.Quessette [1]

[1] *PRiSM, 45, avenue des Etats-Unis,*
Université de Versailles St Quentin en Yvelines,
78000 Versailles, FRANCE

KEY WORDS: Stochastic Bounds, Lumpabillity, Stochastic Monotonicity

ABSTRACT

Despite considerable works, the numerical analysis of large chains remains a difficult problem. Fortunately enough, while modeling computer networks, it is often sufficient to check if the studied mechanism satisfies the requirements of the Quality of Service (QoS) we expect and exact values of the performance indices are not necessary. So we advocate a new method based on the algorithmic derivation of a lumpable stochastic bounding matrix to analyze large state-space problems. Because of the lumpability, the numerical analysis deals with smaller matrices. The stochastic bounds provide upper bounds for performance indices defined as reward functions. Our work is purely algorithmic and algebraic and does not require proofs based on the sample-path theorem and coupling (see [15] for some examples). We present the algorithm, its complexity and memory requirements and an example.

1. Introduction

Our modeling ability has been improved by the tensor representation and composition methods introduced by Plateau [12] and generalized to several high level formalisms (see for instance among many others: Stochastic Petri nets [7]). But our analysis techniques remain roughly the same despite considerable works [14, 17]. Fortunately enough, while modeling high speed networks, protocols or computers, it is often sufficient to check if the proposed mechanism satisfies the requirements of the Quality of Service (QoS) we expect. Exact values of the performance indices are not necessary in this case and bounding some reward functions is often sufficient.

Several methods have been proposed to bound rewards: resolution of a linear algebra problem and polyhedra properties by Courtois and Semal [5] recently adapted to Stochastic Automata

Contract/grant sponsor: Research supported in part by ACI ; contract/grant number: Sure-Paths

Networks by Buchholz [4], Markov Decision Process by Van Dijk [18] and various stochastic bounds (see [15] and references therein). Our method is based on the comparison of sample-paths of Markov chains but it is purely algebraic and algorithmic. Thus the approach we present can be easily included inside software tools based on Markov chains. Unlike former approaches which are either analytical or not enough constructive, this new approach is only based on simple algorithms. These algorithms can always be applied, even if the bounds may be sometimes not enough accurate. These algorithms extend Vincent's algorithm [1] to several aspects : first to insure irreducibility, and mostly to build a matrix easier to solve. Indeed, the matrix obtained by Vincent's algorithm is as difficult as the original one to analyze. Our algorithms provide a way to design an irreducible lumpable upper-bound for an arbitrary partition and a quite general transition matrix. Ordinary Lumpability is an efficient technique to combine with stochastic bounds [2, 16] as a lumped matrix is much simpler to analyze than the original one due to the state space reduction. We also take advantage of some features of the algorithm to avoid to keep the matrix in main memory. In [2], we have presented some applications of this method. Here we present the theorems, the proofs and the algorithms implementation. Let us now review briefly the methodology.

First we have to build on disk a transition matrix P for the problem we want to study. Let n be the size of the state space. n is assumed to be very large (i.e. $n > 10^6$). The storage of P has to be made in a suitable form to help during the bounding process. The main idea is to chose heuristically a partition which implies a large reduction of the state space and may provide an accurate upper bound. We expect that the availability of bounding algorithms associated to numerical softwares will help in the future to design good heuristics. Then the algorithm builds an upper bound R of the transition matrix P. This upper bound is lumpable and the lumped version is stored on disk. The initial matrix P is never in memory and only two vectors of size n are stored in memory during the execution of our algorithms. Thus the main algorithm may even be used to bound a matrix which does not fit in memory. As the lumped matrix size is much smaller, matrix R may be analyzed with usual algorithms. In the typical example we present, the lumped chain has less than 50.000 states and the numerical results already obtained looks accurate enough for our problem.

The following of the paper is organized as follows. First in section 2, we briefly present the stochastic comparison of sample-paths and a sufficient algebraic characterization of the transition matrices of the Markov chains. In this paper we restrict ourselves to Discrete-Time Markov Chains but models in continuous time can be handled after uniformization. We also show in section 2 a basic algorithm (**IMSUB**) which deals with the irreducibility of the bounding matrix. Indeed as the matrix is too large, one cannot check the irreducibility of the bound after it has been computed. One must take advantage of the irreducibility of the initial matrix and design the algorithm to be sure that the resulting matrix is still irreducible. We chose to address the irreducibility problem first to simplify the proofs. Also, Algorithm **IMSUB** may be the root for other algorithms based on various structures for matrices. Section 3 is devoted to the presentation of the real algorithm, its proof, its implementation using only two vectors in memory. Then, in section 4, we present an application of this algorithm to the computation of the loss probabilities in a shared buffer with the Round Robin service discipline and the Pushout memory access algorithm.

2. An algorithmic presentation of stochastic bounds

First, we give a brief overview on stochastic ordering for Discrete-Time Markov chains and we obtain a set of inequalities to imply bounds. Continuous-time chains are handled after uniformization. Then we present a first algorithm derived from the basic one [1]. This first step provides an irreducible matrix. Then we show in the next section how to modify this algorithm to provide a lumpable matrix.

2.1. A brief overview of comparison of Markov chains

In [15], the strong stochastic ordering is defined by the set of non-decreasing functions. Important performance indices such as average population, loss or tail probabilities are non decreasing functions. Therefore, bounds on the distribution imply bounds on these performance indices as well.

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.*

For an algorithmic or algebraic presentation, the following definition is much more convenient. In the following, n will denote the size of matrix P and $P_{i,*}$ will refer to row i of P.

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

The main idea of the sample-path comparison of Markov chains is to prove that the initial ordering of the two distributions at time 0 will be preserved along the path to obtain an upper bound for the steady-state. As we assume that the matrix of the problem does not satisfy any particular property (except irreducibility), all the constraints and the properties must be verified by the upper bounding matrix created by the algorithm. It is known for a long time that monotonicity [11] and comparability of the one step transition probability matrices of time-homogeneous MCs yield these sufficient conditions.

Definition 3 (Comparability) *Let P and R be two stochastic matrices, $P <_{st} R$ iff for all i, we have $P_{i,*} <_{st} R_{i,*}$ (we consider the rows of P and R as vectors).*

Definition 4 (Monotonicity) *Let P be a stochastic matrix, P is st-monotone iff for all u and v, if $u <_{st} v$ then $uP <_{st} vP$.*

Hopefully, st-monotone matrices are completely characterized, and again we obtain some algebraic constraints. The fundamental theorem states that these conditions are sufficient [15]:

Theorem 1 (Stoyan) *Let $X(t)$ and $Y(t)$ two DTMC and P and R their respective stochastic matrices. Then $X(t) <_{st} Y(t)$ if*

- *$X(0) <_{st} Y(0)$,*
- *st-monotonicity of at least one of the matrices holds,*
- *st-comparability of the matrices holds, that is, $P_{i,*} <_{st} R_{i,*}$ $\forall i$.*

Proof: By induction on t:

- Assume that $X(t) <_{st} Y(t)$ (true for $t = 0$)
- As $P <_{st} R$ we get: $X(t)P <_{st} X(t)R$
- Assume R is st-monotone, as $X(t) <_{st} Y(t)$ we have: $X(t)R <_{st} Y(t)R$
- Thus, $X(t)P <_{st} Y(t)R$
- After identification $X(t+1) <_{st} Y(t+1)$

Property 1. *Let P be a stochastic matrix, P is st-monotone iff for all i, $j > i$, we have $P_{i,*} <_{st} P_{j,*}$*

Thus, assuming that P is not monotone, we obtain some inequalities on the elements of R :

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

It must be clear from these relations that it is quite simple to obtain a st-monotone upper bound of P using the degrees of freedom provided by these inequalities. We use them to insure that the bound satisfy some properties useful for the analysis : irreducibility and lumpability (see [2, 8] for other properties and algorithms).

2.2. An Algorithm for an Irreducible Upper-Bound

The simplest idea consists in using a set of equalities, instead of inequalities. These equalities provides, once they have been ordered in increasing order for i and in decreasing order for j in system (2), a constructive way to design a stochastic matrix R which yields a stochastic bound for matrix P: this is Vincent's algorithm [1].

$$\begin{cases} \sum_{k=j}^{n} R_{1,k} & = & \sum_{k=j}^{n} P_{1,k} & \forall\, i,j \\ \sum_{k=j}^{n} R_{i+1,k} & = & max(\sum_{k=j}^{n} R_{i,k}, \sum_{k=j}^{n} P_{i+1,k}) & \forall\, i,j \end{cases} \tag{2}$$

Or, if we assume, as usual, that $\sum_{i}^{j}$ is always equal to zero when $j < i$, and that all the elements of the matrix with index outside $(1..n)$ are all zero, we get a simple relation for all indices i and j:

$$R_{i,j} = \max\left(\sum_{k=j}^{n} R_{i-1,k}, \sum_{k=j}^{n} P_{i,k}\right) - \sum_{k=j+1}^{n} R_{i,k}$$

Vincent and his coauthor have observed that the matrix they computed may be reducible. Indeed, some elements of R may be zero even if the elements with the same indices in P are positive. It is very important to avoid transition deletions because the state space is so large that it is really impossible to check the reducibility of matrix R once it has been computed. We assume that the irreducibility of matrix P is proved using a high level formalism which allows to specify the transition matrix and the reachable states. So it is very important that matrix R obtained by the automatic derivation of the bound is still irreducible.

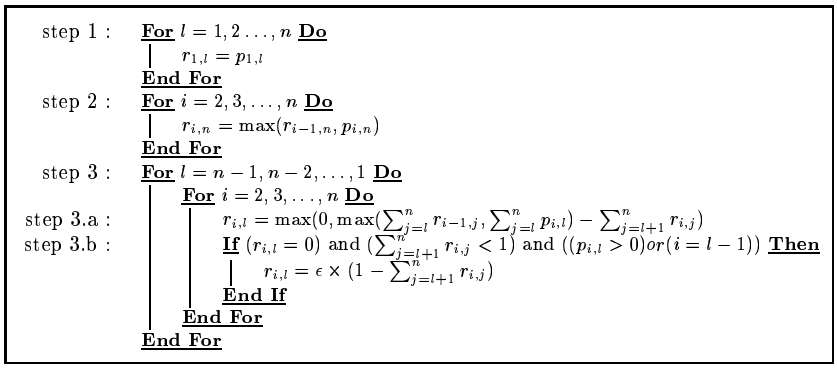

Figure 1. Algorithm **IMSUB**: Construction of an irreducible st-monotone upper bound

We have derived a new algorithm (called **IMSUB**) and we have proved a necessary and sufficient condition on P to obtain an irreducible matrix R. This algorithm is based on three main ideas: respect the basic inequalities (1), avoid the unnecessary deletions of transition and force the elements of the sub-diagonal to be positive. For the sake of simplicity and to emphasis the relations with inequalities (1), we use a matrix representation for P and R and we use the summations $\sum_{j=l}^{n} r_{i-1,j}$ and $\sum_{j=l+1}^{n} r_{i,j}$. We know that they are already computed when we need them due to the the ordering of the indices. Of course the real algorithm presented in section 3.3 uses a sparse matrix version and it only keeps in memory two vectors of size n to store $\sum_{j=l}^{n} r_{i-1,j}$ and $\sum_{j=l+1}^{n} r_{i,j}$.

Theorem 2. *Let P be an irreducible finite stochastic matrix. Matrix R computed from P by Algorithm **IMSUB** is irreducible iff $P_{1,1} > 0$ and every row of the lower triangle of matrix P contains at least one positive element.*

Proof: The assumptions are clearly necessary (see Fig. 2 for the effects of **IMSUB** on the elements). The irreducibility follows from lemma 3. The relations (1) are satisfied, thus the matrix is a monotone upper-bound. Let us begin by two technical lemmas to prove the irreducibility. The key idea of the proof is the following. The main characteristic of the algorithm is based on the monotonicity of R. The indices of the last (the rightmost) non zero elements in rows of R form a non decreasing sequence while it is not true for matrix P. Such a property is also satisfied by the indices of the last (i.e. leftmost) non zero element in the rows of R (see Fig. 2 where the boundaries of non zero elements for P and R are represented by solid lines). These two lines also give a rough boundary for the computations required for R.

Definition 5. *Consider an arbitrary matrix R, let us denote by n_i^R (resp. k_i^R) the first (resp. last) positive element in row i of matrix R.*

Lemma 1. *If $P_{1,1} \neq 0$ and every row of the lower triangle of matrix P contains at least one positive element, then for all $i \geq 2$ we have $n_i^R = \max(n_i^P, n_{i-1}^R)$ and $k_i^R = \max(k_i^P, k_{i-1}^R)$.*

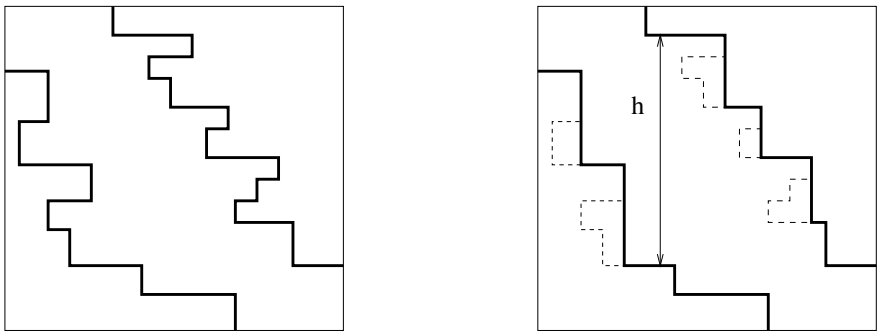

Figure 2. Filling of Matrices P (left) and R (right), the dotted lines in R represent the boundary in P

Proof : As P and R have constant row sum, relations (1) imply that $n_i^R \geq n_i^P$ and $n_i^R \geq n_{i-1}^R$. Therefore $n_i^R \geq \max(n_i^P, n_{i-1}^R)$.

Assume now that $n_i^R > \max(n_i^P, n_{i-1}^R)$. We will show a contradiction. Let us compute R_{i,n_i^R} using step 3.a and consider the two following cases:

- $\max(\sum_{j=n_i^R}^{n} R_{i-1,j}, \sum_{j=n_i^R}^{n} P_{i,j}) - \sum_{j=n_i^R+1}^{n} R_{i,j} \leq 0$. Then we use step 3.b to give a value to R_{i,n_i^R}. This value may be $\epsilon(1 - \sum_{j=n_i^R+1}^{n} R_{i,j})$ or zero, if the test fails. As $\epsilon < 1$, we get $\sum_{j=n_i^R}^{n} R_{i,j} < 1$; a contradiction.
- $\max(\sum_{j=n_i^R}^{n} R_{i-1,j}, \sum_{j=n_i^R}^{n} P_{i,j}) - \sum_{j=n_i^R+1}^{n} R_{i,j} > 0$. Then, Step 3.b is not used and Step 3.a implies that: $\sum_{j=n_i^R}^{n} R_{i,j} = \max(\sum_{j=n_i^R}^{n} R_{i-1,j}, \sum_{j=n_i^R}^{n} P_{i,j})$. As $n_i^R > \max(n_i^P, n_{i-1}^R)$, we get $\sum_{j=n_i^R}^{n} R_{i-1,j} < 1$ and $\sum_{j=n_i^R}^{n} P_{i,j} < 1$. Therefore, $\sum_{j=n_i^R}^{n} R_{i,j} < 1$; again a contradiction.

Thus we have $n_i^R = \max(n_i^P, n_{i-1}^R)$. The proof of the other relation is omitted as it is quite similar to the first one. □

Lemma 2. *If $P_{1,1} \neq 0$ and every row of the lower triangle of matrix P contains at least one positive element, matrix R computed by algorithm **IMSUB** satisfies the same properties. Furthermore, all the transitions in the upper triangle of P still exist in the upper triangle of matrix R.*

Proof: First, Step 1 of the algorithm implies that $R_{1,1} = P_{1,1}$, therefore $R_{1,1} \neq 0$.

Now let us restate the second assumption: $\forall i$ in $(2..n)$, we have $n_i^P < i$. Following lemma 1, we have $n_i^R = \max(n_i^P, n_{i-1}^R)$. As $n_1^R = 1$ and $n_i^P < i$, a simple induction proves that $n_i^R < i$ for all $i > 1$.

Let us now turn to the last property. As $n_i^R < i$ for all $i > 1$, we have $\sum_{j=l}^{n} R_{i,j} < 1$ for all $l \geq i$. Therefore the test is Step 3.b is always true when we consider a transition in the upper diagonal of the matrices. □

Lemma 3. *If P is irreducible and $P_{1,1} > 0$ and every row of the lower triangle of matrix P contains at least one positive element, then matrix R is irreducible.*

Proof: Let x be an arbitrary state, we prove that there is a path in R from 1 to x and from x to 1. By Lemma 2, we proved that every row of the lower triangle of matrix R contains at

least one positive element, there is no destruction of any transition on the upper triangle of matrix R and the elements of the sub-diagonal are positive. Thus there is a path from x to 1 going through the transitions of the sub-diagonal.

Furthermore as P is irreducible, there exists a path from state 1 to state x in matrix P. This path can be divided into 2 kinds of transitions : those in the upper triangle (a transition from a state i to a state j with $i < j$) and those in the lower triangle ($i > j$). Transitions of the first type are not deleted in matrix R. The other type of transitions can be deleted by the algorithm, but they can be replaced by the path from i to j using transitions of the sub-diagonal of R. □

3. An Algorithm for a Lumpable and Irreducible Upper Bound

In section 2, we have addressed the irreducibility problem. Let us now turn to the lumpability constraint. Lumpability implies a state space reduction before the numerical analysis. The algorithms are based on Algorithm **IMSUB** and on the decomposition of the chain into macro-states. Let r be the number of macro-states. Let $A_1..A_r$ the partition of the states we consider. Again we do not assume that P is lumpable according to this partition, but we build algorithmically a lumpable matrix which is also an upper bound of P for the "st" order. First, let us recall the characterization of ordinary lumpable chains [3].

Property 2 (ordinary lumpability) *Let R be the matrix of an irreducible finite DTMC, let A_k be a partition of the states of the chain. The chain is ordinary lumpable according to partition A_k, iff for all states e and f in the same arbitrary macro state A_i, we have $\sum_{j\in A_k} R_{e,j} = \sum_{j\in A_k} R_{f,j}$ for all macro-state A_k.*

3.1. Presentation of the Algorithm

We assume that the states are ordered according to the macro-state partition we consider. This is the main assumption. Let $b(k)$ and $e(k)$ be the indices of the first state and the last state, respectively, of macro-state A_k. Clearly, lumpability constraints are consistent with the st-monotony. The algorithm computes the matrix column after column. Each block needs two steps (see Fig. 3 and Fig. 4). The first step is based on Algorithm IMSUB while second step modifies the first column of the block to satisfy the lumpability constraint.

$$P = \left[\begin{array}{cc|ccc} 0.1 & 0.4 & 0.1 & 0.2 & 0.2 \\ 0.2 & 0.1 & 0.1 & 0.4 & 0.2 \\ 0.1 & 0.6 & 0.2 & 0.1 & 0.0 \\ \hline 0.2 & 0.0 & 0.1 & 0.2 & 0.5 \\ 0.0 & 0.1 & 0.5 & 0.1 & 0.3 \end{array}\right] \tag{3}$$

$$P1 = \left[\begin{array}{c|ccc} & 0.1 & 0.2 & 0.2 \\ & 0.1 & 0.4 & 0.2 \\ & 0.1 & 0.4 & 0.2 \\ \hline & 0.1 & 0.2 & 0.5 \\ & 0.2 & 0.2 & 0.5 \end{array}\right] \quad P2 = \left[\begin{array}{c|ccc} & 0.3 & 0.2 & 0.2 \\ & 0.1 & 0.4 & 0.2 \\ & 0.1 & 0.4 & 0.2 \\ \hline & 0.2 & 0.2 & 0.5 \\ & 0.2 & 0.2 & 0.5 \end{array}\right] \tag{4}$$

More precisely, the first step uses the same relations as Algorithm **IMSUB** but it has to manage the block structure and the fact that the first row of R and P may now be different

due to the second step of the algorithm. The lumpability constraint is only known at the end of the first step. Remember that ordinary lumpability is due to a constant row sum for the block. Thus after the first step, 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 reached for the last row of the block. In step 2, we modify block after block the first column of the block taking into account the last row sum.

Consider matrix P in Eq. 3 to illustrate the behavior of the algorithm. Assume that we divide the state-space into two macro-states: $(1,2)$ and $(3,4,5)$. We show the last three columns after the first step ($P1$ in Eq. 4), and after the second step ($P2$ also in Eq. 4). Remark that, during the second step $P2_{1,3}$ and $P2_{4,3}$ have been changed such that the two blocks now have constant row sum.

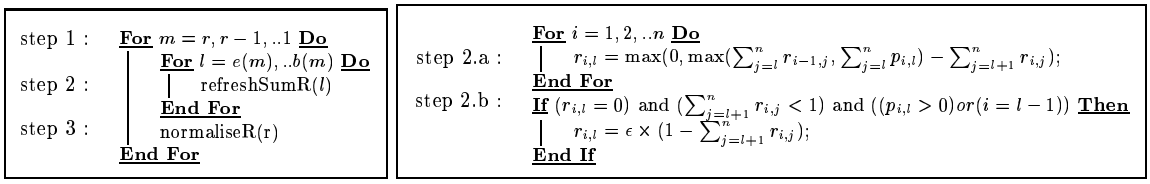

Figure 3. Algorithm **LIMSUB** : Construction of an ordinary lumpable irreducible st-monotone upper bounding DTMC R (left) and method $refreshSumR()$ (right)

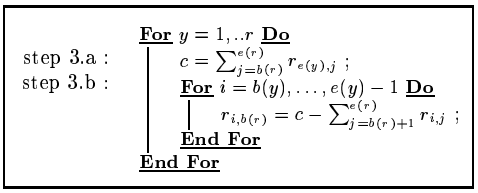

Figure 4. Algorithm **LIMSUB** : method $normaliseR()$

3.2. Proof of the Algorithm

Remember that the irreducibility of a lumpable matrix implies the irreducibility of the lumped version of the matrix [3].

Theorem 3. *Let P be an irreducible finite stochastic matrix, let $A_1 ... A_r$ a partition of the state-space. If the upper left block of P follows the assumptions of lemma 1 and if for all state i in 1..n, there exist a transition to state j such that $i \in A_x$ and $j \in A_y$ and $y < x$, then Algorithm **LIMSUB** computes an irreducible upper bound for the "st" order which is lumpable according to the partition.*

Proof: The irreducibility follows from lemma 5. Relations (1) are clearly satisfied, thus the matrix is a monotone upper-bound. The lumpability is obtained by steps 3.a to 3.b in the

algorithm. Indeed, we compute in Step 3.a the row sum for the last row of the block. As matrix R is st-monotone, the row sum is non decreasing and the maximal row sum is obtained from the last row of the block. Then we modify in Step 3.b the first column to obtain constant row sums. Therefore the matrix is lumpable and is an irreducible upper-bound. □

Lemma 4. *Consider an arbitrary partition $A_1..A_r$. Assume that P follow the assumptions of theorem 3, then:*

- $n_1^R = 1$ *and for all* $i \in A_1$, $i > 1$ *we have:* $n_i^R < i$
- *for all* $i \in A_k$, *with* $k > 1$, *we have:* $n_i^R < b(k) \leq i$

Proof: let z_i^R be the first (the leftmost) positive element in row i of matrix R after Step 2.a and 2.b (before the last normalization in steps 3. As the assumptions of theorem 3 are stronger than the ones used in Lemma 1, and as Steps 2 of Algorithm **LIMSUB** are exactly the same as in Algorithm **IMSUB**, we can apply Lemma 3:

$$z_i^R = \max(n_i^P, z_{i-1}^R) \tag{5}$$

Furthermore we have $z_1^R = 1$ and $z_i^R < i$ for all $i > 2$. But as Steps 3.a to 3.b are not used for A_1, we have: $z_i^R = n_i^R$ for all i in A_1. The first two results are then obtained directly from lemma 3.

The second property is proved by induction on the index of the macro-states. During Steps 3 of algorithm **LIMSUB**, the last elements may be moved to the right. Thus index n_i^R is bigger than z_i^R for all i. But the last row of block A_x is not modified during this steps. Therefore we have for every state index i and block index x: $z_i^R \leq n_i^R$ and $z_{e(x)}^R = n_{e(x)}^R$. As $b(x+1) = e(x)+1$, we obtain: $z_{b(x)}^R = \max(n_{b(x)}^P, n_{e(x-1)}^R)$. Due to the monotonicity, indices n_i^R are not decreasing. Let us now turn to the initial part of the induction (i.e. $x = 2$).

$$z_{b(2)}^R = \max(n_{b(2)}^P, n_{e(1)}^R)$$

The assumptions of theorem 3 show that $n_i^P < b(2)$ for all $i \in A_2$. We also have $n_{e(1)}^R < e(1) < b(2)$. Thus $z_{b(2)}^R < b(2)$. Relation 5 implies that this last inequality is true for all states in A_x. Thus $z_{e(2)}^R < b(2)$.

As $z_{e(2)}^R = n_{e(2)}^R$, clearly we have $n_{e(2)}^R < b(2)$, and $n_i^R \leq n_{e(2)}^R$ for all i in A_2. Finally, $n_i^R < b(2)$ for an arbitrary state in A_2.

The induction from macro state index x to $x + 1$ follows exactly the same scheme and is omitted for the sake of readability. Finally $b(k) < i$ follows from the definition of the block. □

Lemma 5. *If P satisfies the assumptions of theorem 3, then matrix R is irreducible.*

Proof: First, lemma 4 implies that for all state index $i > 1$, we have $n_i^R < i$. Thus the test in Step 2.b is true for elements in the sub-diagonal of R and we know that these elements are positive. Similarly, the positive elements of the sub-diagonal of P are still positive in matrix R even if they have been changed. The end of the proof is the same as for Lemma 3 and is omitted. □

Note that the block by block computation also provides a nice property for matrix Q (see [2] for a proof).

Theorem 4. *Let R be an st-monotone matrix which is an upper bound for P. Assume that R is ordinary lumpable for partition A_k and let $R^{m,l}$ and $P^{m,l}$ be the blocks of transitions from set A_m to set A_l for R and P respectively, then for all m and l, block $R^{m,l}$ is st-monotone*

3.3. Complexity, Memory and Implementation Details

We now present how to implement Algorithm **LIMSUB**, how to minimize the number of vectors in memory and we give some information about the number of operations. We consider a matrix P of size n and a partition of the state-space into r macro-states $A_1, \ldots, A_r$. From the presentation of the algorithm (see again Fig 3), it must be clear that it is possible to implement the column per column version of the algorithm with only one column of matrix P in memory at a time. Thus, we only use two vectors of size n, $sumP$ and $sumR$ to represent respectively $sumP[i] = \sum_{k=j}^{n} P_{i,j}$ and $sumR[i] = \sum_{k=j}^{n} R_{i,j}$. Furthermore, we need a linked list $listIndiceP$ to represent the values and the row indices of non zero elements of column j of P. The value of R is not stored. Instead we store the current column of the lumped matrix ; this requires a vector of size r to store the elements $lumpeeR$ and another vector of the same size to compute them : $sumLumpeeR$ which contains the sum : $\sum_{k=j}^{r} Rl_{[i,j]}$ where Rl is the lumped matrix.

Finally we need to define the partition by the first and last indices for each macro-state (the vectors e() and b() in Algorithm **LIMSUB**). Only one of these vectors is in memory, the other one is a macro definition which simplifies the presentation of the algorithm.

Vector $sumR$ is increasing because R is monotone. Therefore, we use a vector to store $sumR$ but we add two indices for the first non zero elements and the last element smaller than 1. These indices specify the boundary of the loops to avoid unnecessary computations. Let us now turn to the two main methods used in **LIMSUB**: refreshSumR() and normaliseR().

```
addColumnP()
For i = indiceP, ..., indiceEndR Do
    If i == 0 Then
        val = max(0, sumP[0] − sumR[0])
    Else
        val = max(0, max(sumP[i], sumR[i − 1]) − sumR[i])
    End If
    If (val == 0)and(sumR[i] < 1)and(exist(i)or(i == numColumn + 1)) Then
        val = ε · (1 − sumR[i])
    End If
    sumR[i] = sumR[i] + val
End For
```

Figure 5. function $refreshSumR()$

refreshSumR) : This function reads a line in the file containing matrix P in a sparse representation. Each line of this file describes a column of the matrix : the first line corresponds to the last column, the last line to the first column. While reading the line, one stores the indices in array $listIndiceP$, one adds the values in array $sumP$, and one computes $indiceP$ and $indiceR$. Then one proceeds to the main computation (described in figure 5).

Let w_j^P be the index of the first non-zero element of column j in matrix P, Step 2.a of algorithm **LIMSUB** leads to $R_{i,j} = 0$ for $i = 1, \ldots, w_j^P - 1$ and step 2.b is used when $i = j + 1$. Therefore the lower boundary $indiceP$, of the computation loop is the minimum between w and $j + 1$, Similarly, as $sumR$ is non decreasing and smaller than 1.0, it is not necessary to compute the elements once the summation reaches 1.0. They are all equal to zero. We have to compute elements $R_{i,j}$ from $IndiceP$ to $IndiceEndR$ (the index of the last element of $sumR$ smaller than 1).

normaliseR() : For each column, $indiceP$ gives the index from which we have to modify $sumR$. Step 3 fix for all lines i of a same macro-state r $sumR[i] = sumR[e(r)]$.

```
For macro = getMacro(indiceEndRNorm), ..., getMacro(indiceR) Do
    compute e(macro), b(macro)
    max = sumR[e(macro)]
    If max ≠ 0 Then
        For j = e(macro) − 1, ..., b(macro) Do
            sumR[j] = max
        End For   lumpeeR[macro] = max − sumLumpeeR[macro]
        If lumpeeR[macro]! = 0 Then
            nbrElements++
            nbrValues++
            sumLumpeeR[macro] += lumpeeR[macro]
        End If
    Else
        break
    End If
    writeR()
End For
```

Figure 6. function $normaliseR()$

The complexity of algorithm **LIMSUB** is given by the sum for all columns of the number of operations in functions $refreshSumR()$ and $normaliseR()$. This complexity is quite difficult to evaluate because it is dependent on the index of the first non zero element in each column and the index where the summations of elements of R reach 1.0 (a very specific characteristic of our algorithm). Clearly the worst case complexity is $O(n^2)$ but in all the cases we have observed the real complexity of **LIMSUB** is much smaller. Morerover a fundamental result for Algorithms **IMSUB** and **LIMSUB** allows to compute the number of operations before we use the algorithm. Indeed, the numbers of elements of R which are computed is equal to the area between the two solid lines in the right part of Fig. 2. The two edges are given by the induction relation allready obtained. We show this property for algorithm **IMSUB** which is a little bit simpler (the generalization for **LIMSUB** is omitted).

Property 3. *For Algorithm **IMSUB**, the number of operations is in* $O(\sum_{j=1}^{n}(k_i^R - n_i^R))$. *Both sequences are obtained by the induction relations proved in Lemma 1.*

Then for some matrix type, it is possible to get a better estimate of the complexity:

Property 4. *Assume that matrix P has a band structure of size b, then matrix R has the same structure and the number of operations of* ***LIMSUB*** *is $O(n\ b)$*

Note that this implementation of the algorithm does not take into account that P is sparse and that vectors of P are linked lists. So, we have developped a better implementation of this algorithm (LIMSUB 2) which avoids unnecessary computation of zero elements of R. We present in the next section some experimental results for these algorithms.

It is possible to obtain an estimate or an upper-bound of the computation time before the real computation takes place If this number is too large, we have to design a new partition or a new ordering of the macro-states to speed-up the computation. It must be clear that the assumptions of theorem 3 take into account the transition matrix, the partition and the ordering of the states. The ordering must be consistent with the block decomposition and with the rewards which must be a non decreasing function of the state index (because of the "st" majorization). Thus the partition has to be carefully chosen.

Once we have designed a partition and reordered the states according to this partition, one must check the rewards. If this last assumption is not satisfied, we can replace the initial reward function $w(i)$ by another one which will be always larger and which is consistent with the ordering. Let $s(i)$ be this new reward function, it is sufficient to have : $s(1) = w(1)$ and $s(i) = max(w(i), s(i-1))$.

4. Loss Rate in a RR Queue with Pushout Algorithm

We consider a finite queue and a buffer policy which combines the PushOut mechanism for the space management and the Round Robin service discipline. We assume that there exist two types of packets with distinct loss rate requirements. In the sequel, we denote as high priority, the packets which have the highest requirements, i.e., the smallest loss ratio. 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. A low priority packet which arrives in a full buffer is lost. If the buffer is not full, both types of packets are accepted. The buffer size is B. Arrivals of both types of packets follow Poisson processes with rates λ_H and λ_L.

For the sake of simplicity, we assume that both type of packets have the same service time which is exponentially distributed with rate μ. The scheduling algorithm is Round Robin (RR), a simple discipline which exhibits good fairness properties. A simple generalization (Weighted Round Robin) may be an efficient mechanism for Diffserv architectures and Internet. The association of Pushout memory access and Head Of Line service discipline has already been proposed and analyzed for ATM networks [10]. However the analysis is based on approximation whose accuracy is quite impossible to check for large buffers and very small loss probabilities.

4.1. The Model and the Rewards

The description of the state space is multidimensional : (T, H, RR), where T is the total number of packets, H is the number of High priority packets and RR is the scheduler state which shows the type of packets to serve. This representation is unusual for a queue with two types of customers but it is more convenient [8].

Of course we have $H \leq T$ and the number of reachable states is $(B+1)(B+2)$ Clearly we

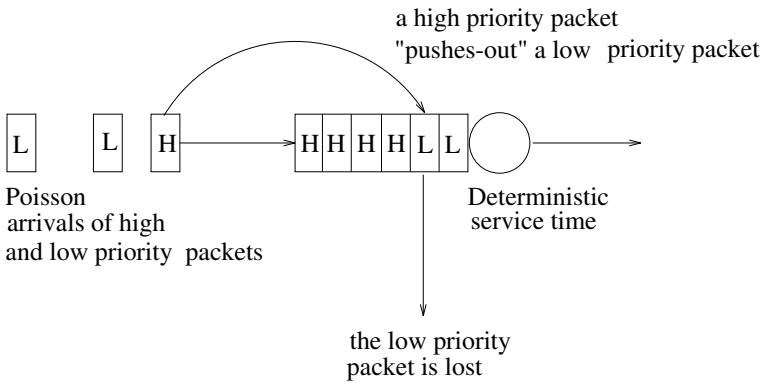

Figure 7. The Pushout mechanism

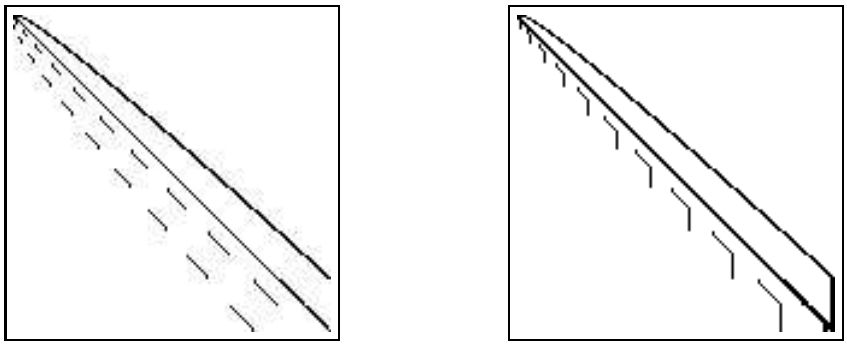

Figure 8. Non zero elements in the initial matrix (left) and in the bound (right) given by **IMSUB**

have a continuous-time Markov chain with at most three transitions for every state (arrival of a high priority cell, arrival of a low priority cell, service of a cell). We want to compute the probability of loss for high priority packets. Due to the Pushout, an arriving packet is lost if the buffer is full of high priority packets. Therefore we want to compute $\pi(B,B,l)+\pi(B,B,h)$.

First we must perform an uniformization to obtain a discrete-time chain. We use a non zero uniformization factor. Thus, the uniformization process adds a new transition for every state in the chain: a loop. Then, the maximal degree is 4. The possible transitions for a RR queue with Pushout are gathered into table I. As usual, $X++$ (resp. $X--$) means incrementation (resp. decrementation) of X.

Note that when the queue is empty (i.e. $T=0$), it is not necessary to represent explicitly the state of the scheduler (i.e. H or L). However we keep both states in the model and we add the last transition (i.e. list processing). This transition states that when the queue is empty we continue to toggle the scheduling state. This transition is exponential with rate μ (i.e. like the service rate). This transition has no effect on the performance of the system as the arrival of the first packet will change again the scheduler. These states and transitions are necessary to state the irreducibility of the bounding matrix (see property 5).

Type	Rate	Transition	Condition
arrival L	λ_L	$T++$	$T < B$
arrival H without pushout	λ_H	$T++$, $H++$	$T < B$
arrival H with pushout	λ_H	$H++$	$T = B$ and $H < B$
service L	μ	$T--$, $Togle(RR)$	$T > 0$ and $(((RR = l)$ and $(T > H))$ or $((RR = h)$ and$(H = 0)))$
service H	μ	$T--$, $H--$, $Toggle(RR)$	$T > 0$ and $(((RR = h)$ and $(H > 0))$ or $((RR = l)$ and$(T = H)))$
list processing	μ	$Toggle(RR)$	$T = 0$

Table I. List of transitions

4.2. Applying Algorithm ***LIMSUB***

First we have to design a partition of the state space. Remember that, as the states in the partition are assumed to be consecutive, this partition step may include a renumbering of the state space. Furthermore, the rewards must be not decreasing. We have considered several partitions but for the sake of conciness we only present one of them, whose results are quite good.

Let us consider a sequence of positive integers F_i. Let us denote by $\mathcal{F}$ the partition we now design. We define macro-states (T, Y, RR) where $Y = max(H, T - F_T)$.

- If $Y > T - F_T$, then the macro-state contains only one state (T, H, RR) and Y is exactly the number of high priority packets in the buffer.
- If $Y = T - F_T$, then the state (T, Y, RR) is a macro-state which contains all the states (T, X, RR) such that $X \leq T - F_T$.

Note that the states of the initial chain where packets are lost (i.e. (B, B, h) and (B, B, l)) are not aggregated into macro-states as $F_B > 0$.

The states are ordered according to the lexicographic ordering on the states (T, Y, RR). Again, this ordering has very important properties one must check. First, we want to compute the probability that an incoming high priority packet is rejected. Due to PASTA, this probability is also the probability that the buffer is full of high priority packets (i.e. $T = B$ and $H = B$). Thus, the reward is equal to zero except for states (B, B, h) and (B, B, l) where its value is 1. As these states are the last states in the lexicographic ordering, the reward is clearly a non decreasing function.

Property 5. *The chain, the ordering and partition $\mathcal{F}$ satisfy the assumptions of theorem 3.*

Proof: Indeed, the chain is clearly irreducible (see the list of transitions in the former table). Due to the uniformisation procedure, we clearly have $P(1, 1) > 0$. The first states are $(0, 0, h)$ and $(0, 0, l)$. Due to the "list processing" transition, we have transitions between theses two states. Thus we have $P(2, 1) > 0$. Furthermore if T is positive, there exists always a transition from an arbitrary state i with T packets to a state j with $T - 1$ packets. According to the partition definition and ordering, the index of macro-state which contains i is strictly larger than the index of macro-state which contains j. Thus the matrix satisfies the assumptions of theorem 3.

Let us now present some typical computation times and sizes of models. We show in the next two tables the computation times for the various tasks we have to do. All the programs were written in *C*, compiled with *gcc* at the maximal optimization level and executed on an usual PC with a 1.7GhZ processor.

B	Size	NZT	Generate	Bound Size	Bound NZT	Reorder and Store
500	251502	1005003	3.3s	11872	79550	5.8s
1000	1003002	4009995	13.s	23850	148371	23.2s

Table II. Sizes of for exemples

In both tables, B is the buffer size. Then we report the size of the original matrix: number of states and number of non zero transitions (NZT) and the time to build this matrix and store it on disk. Then, we give the size of the bounding matrix (the F_i are all equal to 10 in these experiments) and the number of non zero transitions. To compute the bound, we need some pre-processing steps: first reorder the states according to the partition, then store the matrix in a suitable form. The last column reports the time for preprocessing the matrix (i.e. reordering and storing) required to apply **LIMSUB**.

We also report in table III the computation times for the algorithms we have presented before. **LIMSUB** is the basic algorithm while **LIMSUB 2** is a more efficient implementation which avoids the computation of some null entries of the matrices. Finally, for the sake of comparison, we also give the computation times for **IMSUB** algorithm and for numerical solution. This last step is performed by a standard Gauss-Seidel algorithm. Clearly, the actual implementation of **LIMSUB** has to be optimized and **LIMSUB 2** is much more efficient.

B	**LIMSUB**	**LIMSUB 2**	**IMSUB**	Resolution
500	44.3s	21.8s	10.1s	36.3s
1000	272s	86.7s	38.2s	73.6s

Table III. Sizes of for exemples

Numerical results can be found in [2, 8, 9] where comparisons with exact results are obtained when the chain is small. It can be observed that the bounds are good when the steady-state distributions are skewed with large probabilities for states with small values for T. We have also observed [8] that this state representation and this ordering imply that matrix P is almost "st"-monotone. Thus, we only need a small number of perturbations to obtain a lumpable bound.

5. Conclusions

We design a new algorithm to obtain a matrix simpler to solve than the stochastic matrix of a Markov chain. Clearly the performance of **LIMSUB** are dependent of the ordering of the macro-states. It is clearly understood for the accuracy but it is also true for the number of operations for the computation of R. Hopefully the number of operations is easily numerically computed before we really build the bound. Thus one can now study good heuristics for the

ordering. The availability of this algorithm (the program will be on the web soon) will help to study the design of heuristics for accurate bounds.

REFERENCES

1. O. Abu-Amsha, J.-M. Vincent, "An algorithm to bound functionals of Markov chains with large state space", 4th INFORMS Conference on Telecommunications, Boca Raton, Florida, 1998.
2. M. Benmammoun, J.M. Fourneau, N. Pekergin, A. Troubnikoff, "An algorithmic and numerical approach to bound the performance of high speed networks", IEEE MASCOTS 2002, Fort Worth, USA, pp. 375-382.
3. P. Buchholz, "Exact and Ordinary Lumpability in Finite Markov Chains", J. Appl. Prob., V31, pp 59-75, 1994.
4. P. Buchholz, "An iterative bounding method for Stochastic Automata Networks", Performance Evaluation, V49, 2002, pp. 211-226.
5. P.J. Courtois and P. Semal, "Computable bounds for conditional steady-state probabilities in large Markov chains and queueing models" *IEEE JSAC*, 4(6), 1986.
6. T. Dayar, W. J. Stewart, "Comparison of partitioning techniques for two-level iterative solvers on large sparse Markov chains", SIAM Journal on Scientific Computing 21 (2000), pp. 1691-1705.
7. S. Donatelli, "Superposed generalized stochastic Petri nets: definition and efficient solution", Proc. 15th Int. Conf. on Application and Theory of Petri Nets, Zaragoza, Spain, June 1994.
8. 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.
9. J.M. Fourneau, M. Le Coz, N. Pekergin, F. Quessette, "An open tool to compute stochastic bounds on steady-state distributions and rewards", IEEE Mascots 03, USA, 2003.
10. G. Hébuterne and A. Gravey, "A space priority queueing mechanism for multiplexing ATM channels", ITC Specialist Seminar, Computer Network and ISDN Systems, Vol. 20 (Dec. 90), 37–43.
11. J. Keilson, A. Kester, "Monotone matrices and monotone Markov processes", Stochastic Processes and Their Applications 5 (1977), pp. 231-241.
12. B. Plateau, "On the stochastic structure of parallelism and synchronization models for distributed algorithms", Proc. of the SIGMETRICS Conference, Texas, 1985, pp. 147-154.
13. W.J. Stewart, "Introduction to the Numerical Solution of Markov Chains", Princeton Univ. Press, 1994.
14. W.J. Stewart, K. Atif, B. Plateau, "The numerical solution of stochastic automata networks", European Journal of Operational Research 86 (1995) pp. 503-525.
15. D. Stoyan, "Comparison Methods for Queues and Other Stochastic Models", Wiley, 1983.
16. L. Truffet, "Reduction Technique For Discrete Time Markov Chains on Totally Ordered State Space Using Stochastic Comparisons", Journal of Applied Probability, V37, N3, 2000.
17. E. Uysal, T. Dayar, "Iterative methods based on splittings for stochastic automata networks", European Journal of Operational Research, V110 (1998), pp. 166-186.
18. N. Van Dijk, "Error bound analysis for queueing networks", Performance 96 Tutorials, Lausanne, 1996.