Computer Science Department Technical Report
University of California
Los Angeles, CA 90024-1596

BOUNDS FOR QUASI-LUMPABLE MARKOV CHAINS

G. Franceschinis
R. Muntz

February 1993
CSD-930003

Bounds for Quasi-lumpable Markov Chains

Giuliana Franceschinis
Università di Torino
Dipartimento di Informatica

Richard R. Muntz
University of California at Los Angeles
Computer Science Department

Abstract

In this paper a method is proposed to compute bounds on the performance of systems that can be described by *quasi-lumpable* Markov chains (MC). The aim of the work is to cope with the state-space explosion problem using a state aggregation approach. Informally, a quasi-lumpable MC is one which can be made lumpable by relatively small perturbation to the transition rates. The technique for the computation of bounds is based on the Courtois and Semal's method presented in [3, 2], and in particular it can be regarded as a special application of the bounded aggregation method. A technique to improve the bounds is discussed which can be applied whenever bounded aggregation is used. Two examples are included, and the potential computational advantages of the method are discussed.

1 Introduction

The work presented in this paper is related to the lumpability approach to state space reduction. It focuses on Markov chains that incorporate some structural regularity of the kind required by lumpability methods. It can be regarded as a generalization of the lumpability method since it applies to a larger set of models, and gives the exact values for the desired performance figures when the model is actually lumpable, while it gives upper and lower bounds on these figures when the model is not exactly lumpable. In the paper we define the concept of a *quasi-lumpable* Markov chain and show that given a *quasi-lumpable* Markov chain, it can be altered in such a way that the new resulting MC is lumpable and is such that Courtois and Semal's steady state probability bounding methods [3, 2] can be applied to the new lumped MC to obtain bounds on the performance measures of the original model.

In Section 2 the definition of quasi-lumpable MC is given, and some issues about the extent of applicability of the method are discussed. The most important issue addressed in this section is how to check if a given system has a quasi-lumpable underlying MC. We claim that it is not reasonable to try to detect the quasi-lumpability property by inspecting the MC itself, but rather by exploiting some high level property of the abstract model from which the MC is derived. A discussion about the high level formalism of Stochastic Well Formed Nets (SWN) and its relation with quasi-lumpability is included.

Section 3 is the core part of the paper and contains the theoretical details concerning the MC alteration, lumping and solution phases. The method is also reinterpreted as a particular application of Courtois and Semal's Bounded Aggregation method [2]. In the same section there follows a discussion on the computational advantages of the proposed method. The possibility of deriving the lumped MC directly, without building the complete one first, would make the method more effective. We will briefly discuss how this goal can be achieved when some specific high level formalism is used to describe the model.

In Section 4 an original technique is described for further refinement of the computed bounds. The technique is not tied to the specific type of models considered in this paper, and can be always applied within the framework of bounded aggregation methods.

In Section 5 an application example is described and discussed. Finally in Section 6 we summarize the results presented in this paper and discuss some possible directions for future research.

2 Quasi-Lumpable Markov Chains

Definition 1 *A Deterministic Time Markov Chain is said to be* ϵ-quasi-lumpable *with respect to a given state space partition* $\mathcal{A}$ *if its transition probability matrix* P *can be rewritten as* $P = P^- + P^\epsilon$ *, where* P^- *is the largest lower bound (componentwise) for* P *that satisfies the following lumpability conditions [5]:*

$$\forall a_i, a_j \in \mathcal{A}, \forall s \in a_i : \sum_{s' \in a_j} P^-[s, s'] = K_{i,j}$$

($K_{i,j}$ *is a constant value that depends only on* i *and* j*), and no element in* P^ϵ *is greater than* ϵ *in value. A similar definition applies to CTMC, where the probability matrix* P *is substituted with the infinitesimal generator* Q*.*

The intention is that P^ϵ is a matrix with many more zero elements than P^- and with relatively small non-zero elements. Henceforth we use the term "quasi-lumpable DTMC" for a DTMC that is ϵ-quasi-lumpable for some ϵ.

An example[1] of P, P^- and P^ϵ is given in Figure 1 (we assume that $\lambda_2 = \lambda_1 + \epsilon$). The state partition is the following:

$$\mathcal{A} = \{\{1\}, \{2, 3\}, \{4, 5\}, \{6\}\}$$

Quasi Lumpable MC and High Level Modeling Formalisms The motivation for this work on quasi-lumpable MC came from the observation that there are systems that happen to have a very regular state space due to their highly symmetric structure. Often the regularity allows the exact lumping of the state space, while other times the lumping is prevented by some "small" perturbation in the transition rates.

[1]The diagonal elements of P are such that the matrix is stochastic.

Some work exists that takes advantage of a high level description of the system, in which the structural symmetries are evident, to automatically generate a lumped state space [7, 1]. In this section we will explain through an example that this approach can be extended to the cases in which the state space has only a quasi-lumpable structure; we'll specifically refer to the high level formalism of Stochastic Well-formed Nets (SWN) [1]. SWN is a variation of the Colored Petri Net formalism [4] with a special syntax for both color classes and arc function description. The transition instances can either have an associated exponential firing delay or be immediate, meaning that they fire in zero time. Graphically we represent timed transitions as rectangular boxes and immediate transitions as bars.

Usually, a state space generated from a high level model of a system is lumpable when the chosen state representation is so detailed that the behavior description incorporated in the state space is redundant. In this case lumping is equivalent to choosing a more abstract state representation in which many states that were distinct with respect to the more detailed representation become undistinguishable.

A classical example for different possible state description detail levels is the following. Consider the closed system depicted in Figure 2.(a) and assume that there are 5 customers in the system. Customers cycle between two service centers. The first one is a single server machine (the associated delay is an exponentially distributed random variable with parameter μ). The second is a multiple server machine with four servers. A customer chooses randomly one of the four servers on each visit to the multiserver node. Let's assume that the choice is equiprobable for all queues. We also assume that the service time at each server is exponential and does not depend on the customer identity. Of course the state description chosen depends on the distribution of service times and on the scheduling policy of each server, moreover it might also depend on what performance measure we need. In Figure 2.(b) a SWN representation of the system is depicted.

A quite natural representation for the state of this system (suggested by the queueing network model) is a five element vector whose i^{th} element represents the number of clients in the i^{th} queue. Let us call this state representation, $R1$. However, as long as the service rates of the four servers that work in parallel ($S1 - S4$) are the same (e.g., λ), a more abstract representation $R2$ for the state can be used, namely $\langle n_0, nq_0, nq_1, \ldots, nq_5 \rangle$ where n_0 is the number of customers in the first queue, and nq_j, $j = 0, \ldots, 5$, is the number of queues of length j in the second stage of four servers. Hence, the state $\langle 1, 1, 2, 1, 0, 0, 0 \rangle$ of type $R2$

$$P = \begin{array}{|c|c|cc|cc|c|}
\hline
 & 1 & 2 & 3 & 4 & 5 & 6 \\
\hline
1 & \bullet & 2\lambda_1 & \lambda_2 & & & \\
\hline
2 & & \bullet & & \lambda_1 & \lambda_2 & \\
3 & & & \bullet & & 2\lambda_1 & \\
\hline
4 & & & & \bullet & & \lambda_2 \\
5 & & & & & \bullet & \lambda_1 \\
\hline
6 & \mu & & & & & \bullet \\
\hline
\end{array}
=
\begin{array}{|c|c|cc|cc|c|}
\hline
 & 1 & 2 & 3 & 4 & 5 & 6 \\
\hline
1 & \bullet & 2\lambda_1 & \lambda_2 & & & \\
\hline
2 & & \bullet & & \lambda_1 & \lambda_1 & \\
3 & & & \bullet & & 2\lambda_1 & \\
\hline
4 & & & & \bullet & & \lambda_1 \\
5 & & & & & \bullet & \lambda_1 \\
\hline
6 & \mu & & & & & \bullet \\
\hline
\end{array}
+
\begin{array}{|c|c|cc|cc|c|}
\hline
 & 1 & 2 & 3 & 4 & 5 & 6 \\
\hline
1 & & & & & & \\
\hline
2 & & & & & \epsilon & \\
3 & & & & & & \\
\hline
4 & & & & & & \epsilon \\
5 & & & & & & \\
\hline
6 & & & & & & \\
\hline
\end{array}$$

Figure 1: An example of P, P^-, P^ϵ.

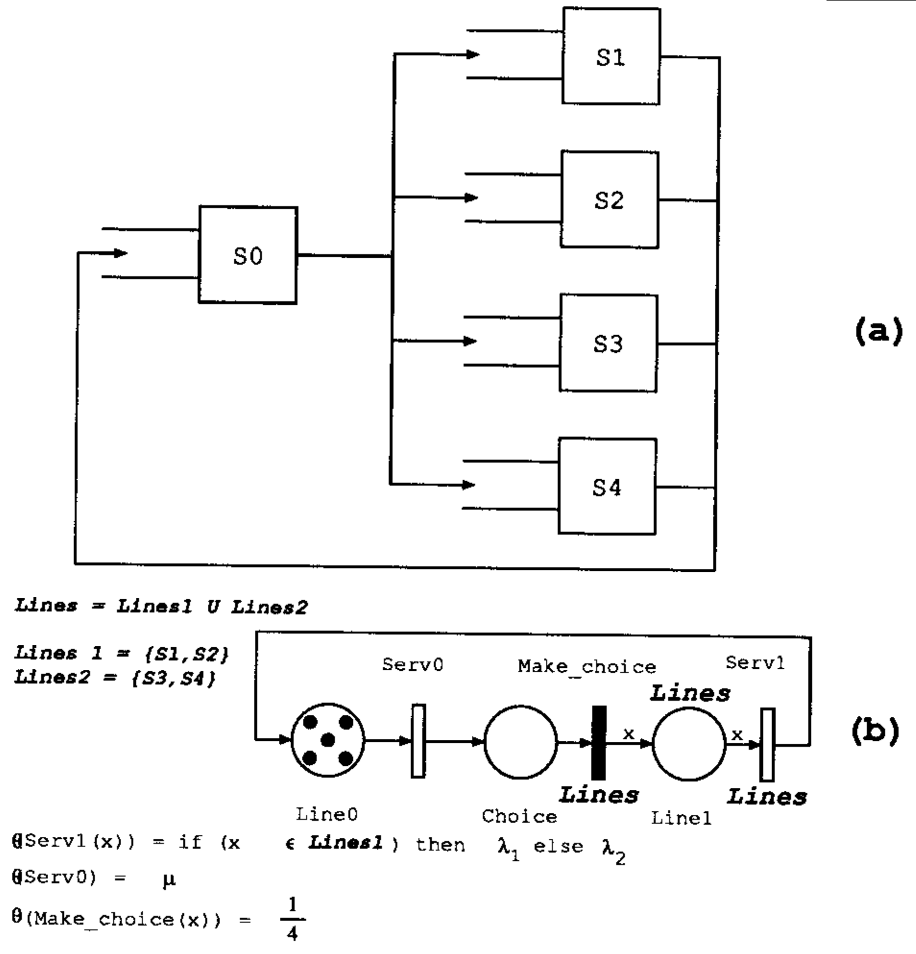

Figure 2: A Multiple Server System Model.

comprises several (more detailed) states of type $R1$, for example $\langle 1,2,1,1,0\rangle$ and $\langle 1,1,0,1,2\rangle$.

In case the four servers in parallel don't have the same speed, for example two of them have a given speed λ_1 while the other two have speed $\lambda_2 > \lambda_1$, then the first description is still too detailed while the second is too abstract. The appropriate state representation (called $R3$) in this case is $\langle n_0, nq_0^1, \ldots, nq_5^1, nq_0^2, \ldots, nq_5^2\rangle$ where n_0 is still the number of customers in the first queue while nq_j^i is the number of queues of length j among the servers with speed λ_i.

Observe that if the detailed state description $R1$ is used, then the structure[2] of the state transition graph for the system where $S1 - S4$ have the same speed is equal to that of the system where two servers have speed λ_1 and the other two have speed λ_2. However if the four parallel servers do not have exactly the same service time then the aggregation can only be partial because some of the state transitions cannot be merged due to their different associated rates. If the difference in the service time of the two groups of servers is relatively

[2]Two state transition graphs have the same structure iff there is a bijection γ between the two set of states $\mathcal{S}_1$ and $\mathcal{S}_2$, and the set of non-negative state transitions $\mathcal{T}_1 = \{(s,s')\}$ of $\mathcal{S}_1$ and $\mathcal{T}_2 = \{(q,q')\}$ of $\mathcal{S}_2$ are such that $\mathcal{T}_2 = \{(q,q') \; : \; q = \gamma(s) \wedge q' = \gamma(s') \wedge (s,s') \in \mathcal{T}_1\}$.

small, then we will call the system quasi-lumpable. So quasi-lumpability is a property that can be observed in systems where a regularity in the state space structure is only partially reflected in the transition rates. Very often in this situation the modeler might decide to introduce an approximation and force the transition rates to become equal, thus forcing the model to become lumpable, using some kind of weighted average for the modified transition rates. This approach however may produce misleading results. The method proposed in the next section is safer from this point of view since it gives bounds rather than an approximation of uncertain error.

In the literature a number of works exist dealing with the automatic construction of a lumped MC from a high level description of the model. Some of these methods could be extended using the bounding method proposed in this paper; for example such extension is possible for the formalism of Stochastic Well-Formed Nets with the associated Symbolic Reachability Graph lumping method.

The Symbolic Reachability Graph (SRG) approach for SWNs relies on the possibility to automatically discover and exploit possible symmetries in the state transition graph (reachability graph): indeed from the SWN representation of the above system the SRG generation algorithm automatically chooses the correct level of state description detail. This is done by defining the color class of servers (*Lines*), with one color for each of the four parallel servers in the second stage. The clients flowing in the system are represented by tokens in the places. The indistinguishable tokens in place $Line0$ represent customers in the first server. The tokens in place $Line1$ represent customers in the four parallel servers and take the color that corresponds to the queue that they join. The natural detailed state description for a state in this net is given by the distribution of tokens (with associated color) in the places. So for example the state $\langle 1,2,1,1,0\rangle$ would be represented as $Line0(1)\ Line1(2<S1>,1<S2>,1<S3>)$. Clearly this state representation is equivalent to the detailed one, previously defined for the queueing description[3]. The evolution of the net among states is performed by transition firing. Transitions are associated with events in the system. So $Serv0$ is associated with the completion of a service by the first server. Transitions *choice* and $Serv1$ are "parametric"; the parameter (x) must be instantiated to a value from *Lines* before the transition fires. The event associated with $choice(Si)$ is the choice of joining the queue of server Si, $i = 1,\dots,4$ after exiting $S0$ (this event is supposed to take no time). The event associated with $Serv1(S_i)$ is the completion of a service by S_i.

A peculiar aspect of SWNs is that the following equivalence relation among markings is always true, independently of the model:

$$M \sim M' \text{ if } \exists s,\ s.t.\ s(M) = M'$$

where s is a set of permutations within the color classes. For example the two states $Line0(1)\ Line1(2<S1>,1<S2>,1<S3>)$ and $Line0(1)\ Line1(1<S1>,1<S3>,2<S4>)$ are equivalent since there exists a permutation s on the objects of class

[3]Place *Choice* can be ignored in the state description because tokens spend no time in it since transition *Make_choice* takes no time to fire; the place simply represents the choice point where a neutral token decides which server color to take before entering place $Line1$.

Lines, namely the permutation that exchanges $S2$ with $S4$, that applied to the first marking returns the second one. The equivalence class to which the two above markings belong, could be described as the set of markings that have one client in service at the first single server station, and four clients in the multiserver station, three of them are being served, while the fourth is queued to one of the three busy servers. The equivalence classes induced by this relation are the state aggregates to be used in the lumping process. The justification for this automatic state grouping is due to the fact that objects in the same class (the servers class *Lines* in our examples) will surely behave homogeneously due to the definition of SWN dynamics. Color classes can be partitioned into subclasses when it is necessary to specify different behavior of similar objects (for example *Lines* is partitioned in $Lines_1$ and $Lines_2$ to distinguish between fast and slow servers). When this happens, the permutations used to define the equivalence relation are restricted to permute objects within the same subclass.

In our example with different speed servers, the servers class must be partitioned into two subclasses to distinguish between the faster and slower servers. This is enough to properly restrict the extent of automatic state aggregation. However, it might be useful to distinguish the case in which a partition into subclasses is used only to specify different transition rates and identical token flow from the case in which the partition into subclasses is used to specify differences in the token flow. In fact in the former case the underlying MC has a quasi-lumpable structure.

The bounding method also applies to systems were the departure/arrival rates are state dependent and are a smooth function of the system population. In this case an aggregate is naturally defined as a set of states whose associated population is within a given range.

When a reward process is associated with the CTMC, then the reward function definition should also be taken into account when checking lumpability (see [6]). A sufficient (though strong) condition is that the reward function gives the same value for every state in an aggregate. We'll see how this condition can be relaxed in our framework of bounding.

3 Bounds for Quasi-lumpable Markov Chains

In this section we first summarize the Courtois and Semal's results that we are going to use; then we show how a quasi-lumpable MC can be transformed into a lumpable one and how Courtois and Semal's methods can be applied to it.

3.1 The Courtois and Semal Bounding Methods

In this section we summarize Courtois and Semal's results [2, 3] that are relevant to our discussion. The results reported here are specific to stochastic matrices, and are thus less general than those presented in [2, 3]; the reader can refer to the original papers for the more general theorems and their proofs (which are for non-negative matrices).

Let P be a stochastic matrix, and let P^- and P^+ be a lower and upper bound (componentwise) for P respectively. We assume that P, P^- and P^+ are all irreducible.

Lemma 1 *[2]*

- *$(I - P^-)$ is non-singular*
- $(I - P^-)^{-1} \geq 0$
- $(I - P^-)^{-1}\mathbf{e}^{\,T} > 0$

where $\mathbf{e}^{\,T}$ *denotes a properly dimensioned column of ones.*

Theorem 1 *[2] Let P be a stochastic matrix, and let P^- be a lower bound for P. We assume that both P and P^- are irreducible. The steady state probability π of P belongs to the polyhedron whose vertices are the rows $\{z_j\}$ of the matrix Z with indices in the set $J = \{j \in 1, \ldots, n,\ s.t.\ \exists i : P_{i,j} > P^-{}_{i,j}\}$*
where

$$Z = \Omega^{-1}(I - P^-)^{-1}$$

and Ω is a diagonal normalization matrix defined as follows:

$$\Omega = diag((I - P^-)^{-1}\mathbf{e}^{\,T})$$

Hence there exists a vector β s.t.

- $\beta\ \mathbf{e}^{\,T} = 1$
- $\beta_j = 0,\ j \notin J$
- $\pi = \beta\ Z$

and

$$\forall i,\ \pi_i^- \leq \pi_i \leq \pi_i^+$$

where

$$\pi_i^- = max\{min_{k\in J}(Z)_{k,i}; 1 - \sum_{j\neq i} max_{k\in J}(Z)_{k,j}\}$$

and

$$\pi_i^+ = min\{max_{k\in J}(Z)_{k,i}; 1 - \sum_{j\neq i} min_{k\in J}(Z)_{k,j}\}$$

The following theorem gives an alternative way for obtaining bounds without computing inverses.

Theorem 2 *[2] Given a lower bound P^- for P, the k^{th} row of matrix Z is the steady state probability vector of the matrix obtained by incrementing the k^{th} column of P^- in order to make it stochastic.*

Thus if we have only a lower bound P^- of the transition probability matrix of a given DTMC (let's assume that the chain has n states), we can still compute bounds for its steady state probability vector π. This is done by computing the solution of n DTMCs; the transition probability matrix P_i of the i^{th} DTMC is obtained from the substochastic matrix P^- by increasing the elements of column i enough to make P_i stochastic. Let π^i be the steady state probability vector solution of the i^{th} DTMC. The lower (upper) bound on the steady state probability of state k is computed as $min_i\ \pi^i_k$ $(max_i\ \pi^i_k)$.

The next theorem is analogous to Theorem 1 when an upper bound P^+ of P is available instead of a lower bound. However this result is slightly weaker because the inverse is not guaranteed to exist; furthermore, even when it does exist some additional constraints have to be tested in order to derive bounds from it (i.e., there is no result similar to Lemma 1 in this case).

Theorem 3 *[2] Let P be a stochastic matrix, and let P^+ be an upper bound for P. We assume that both P and P^+ are irreducible. If the following inverse matrix*

$$(I - P^+)^{-1}$$

exists and satisfies

$$(I - P^+)^{-1}\mathbf{e}^{\,T} < \mathbf{0}^{\,T} \quad \mathbf{e}\,(I - P^+)^{-1} < \mathbf{0}$$

then there exists a vector β s.t.

- $\beta \mathbf{e}^{\,T} = 1$
- $\pi = \beta \Omega^{-1} (I - P^+)^{-1}$

where Ω^{-1} is a diagonal normalization matrix.

In [2, 3] two applications of the above theorems are presented.

1. Computation of bounds on conditional steady state probability of a subset of states $\mathcal{S}$ in a DTMC when the transition probability among the states in the subset is known while only partial or no information is available about the transition probability between states in $\bar{\mathcal{S}}$ and states in $\mathcal{S}$. Of course the amount of information available affects the accuracy of bounds.

   The less accurate and at the same time less computationally expensive way of obtaining these bounds is to applying Theorem 1 using the submatrix $P_{\mathcal{S},\mathcal{S}}$ of transition probabilities among states in $\mathcal{S}$ as P^-.

2. Computation of bounds on the steady state probability vector of a large system by decomposition into smaller subsystems: this is called the *bounded aggregation method.* This method consists of two steps:

   - computation of bounds on the conditional state probabilities within each aggregate using the method just explained;
   - computation of bounds on the probability of being in each aggregate. This part is divided into two steps: (1) derivation of a lower bound $\mathcal{P}^-$ for the inter-aggregate probability matrix $\mathcal{P}$ using the results of previous step; (2) application of Theorem 1 to $\mathcal{P}^-$ to compute bounds on the aggregates probability vector.

Observe that if we knew the vector of conditional probabilities $\pi^{\mathcal{S}}$ of the states in aggregate $\mathcal{S}$, then we could easily compute the transition probability $\mathcal{P}_{\mathcal{S},\mathcal{S}'}$ between aggregates $\mathcal{S}$ and $\mathcal{S}'$ as $\sum_{s,s':s\in\mathcal{S},s'\in\mathcal{S}'} \pi_s^{\mathcal{S}} P_{s,s'}$. Since we know only bounds on $\pi^{\mathcal{S}}, \forall \mathcal{S}$, we are able to compute only a bound $\mathcal{P}^-$ for $\mathcal{P}$.

These techniques can be used to compute bounds not only on the steady state probability vector, but also on any kind of a performance metric defined in terms of a reward function r from the set of states to $\mathbb{R}$. Let r_j denote the reward associated with state j, and Let π^i denote the steady state probability vector obtained by solving the DTMC characterized by the matrix P_i obtained from P^- by increasing its i^{th} column such as to make it stochastic. The mean accumulated reward $\bar{R}$ is defined as

$$\bar{R} = \pi r^T = \sum_i \pi_i r_i$$

and can be rewritten as

$$\bar{R} = \sum_i (\sum_j \beta_j \pi_i^j) r_i = \sum_j \beta_j \sum_i \pi_i^j r_i.$$

The term $\sum_i \pi_i^j r_i$ is equal to the mean accumulated reward $\bar{R}_j = \pi^j r^T$ computed from $P^-{}_j$ and

$$\bar{R} = \sum_i \beta_i \bar{R}_i$$

so that $min_i \bar{R}_i$ and $max_i \bar{R}_i$ are a lower and an upper bound for $\bar{R}$ respectively.

3.2 The proposed method

Transforming a Quasi-Lumpable MC into a lumpable MC The definition of a quasi-lumpable MC contains the first hint on how to proceed in transforming the quasi-lumpable chain into a lumpable one: with reference to matrix P in Figure 1, if the values of ϵ were small enough we could be tempted to just use $P^- + diag(P^\epsilon \mathrm{e}\ ^T)$ instead of P and obtain an approximate result from the lumped version of this new matrix.

Even if this method leads to a good approximation in some cases, in general it is difficult to forecast the actual influence of the approximation on the result both from a quantitative point of view (how far is the approximation from the actual result?) and a qualitative point of view (is the approximation an upper or a lower bound with respect to the correct result?).

Instead of just "forgetting" about the elements in P^ϵ, we could imagine redirecting these "*disturbing*" transition probabilities (or rates) to a new extra state s. By doing this we end up with a new stochastic matrix P_s that has one more state than P^-. The new matrix is depicted in Figure 3; vectors x and y^T are defined as follows:

- $\mathrm{y}^T = P^\epsilon \mathrm{e}\ ^T = (P - P^-)\mathrm{e}\ ^T$
  where $\mathrm{e}\ ^T$ is a column vector of 1s, of appropriate dimension; y^T is a column vector whose elements are the row sums of P^ϵ (hence the interpretation of element i of vector y^T as the global probability of going from state i to state s due to a redirection of probability);
- x is a row vector of the same dimension as a row of P^-; right now we do not define its contents but we do impose the constraint $\mathrm{xe}\ ^T = 1$.

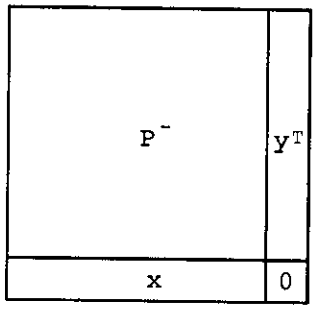

Figure 3: Modified matrix P_s.

We claim that there exists an assignment of values to the elements of $\mathbf{x}$ such that the steady state probability of MC P is equal to the conditional probability of the first n states[4] of chain P_s. In other words, if we redistribute "properly" the transition probabilities from the extra state to the states that have had a decrement in the incoming transition probabilities, we end up with the same conditional probability distribution as in the original model.

Theorem 4 *Let P be the transition probability matrix of an ergodic DTMC with n states. Let P^- be a lower bound (componentwise) for P, i.e., $P^- \leq P$. Let $\mathbf{y}^T = (P - P^-)\mathbf{e}^{\,T}$ and finally let P_s be the stochastic matrix of Figure 3.*

There exists a vector $\mathbf{x}$ such that the conditional steady state probabilities for the first n states of P_s is equal to the steady state probabilities of DTMC P.

Proof: The equations for the computation of the steady state probabilities in the derived chain characterized by the state transition matrix P_s are

$$\begin{cases} (\pi', a)P_s = (\pi', a) \\ (\pi', a)\mathbf{e}^{\,T} = 1 \end{cases} \tag{1}$$

Let's assume that π is the vector satisfying the equations

$$\begin{cases} \pi P = \pi \\ \pi \mathbf{e}^{\,T} = 1 \end{cases}$$

i.e., π is the steady state probability vector of the original problem.

The first equation in system (1) can be rewritten as the system

$$\begin{cases} \pi' P^- + a\mathbf{x} = \pi' \\ \pi' \mathbf{y}^T = a \end{cases}$$

Substituting $\pi'\mathbf{y}^T$ for a in the first equation we get

$$\pi' P^- + \pi' \mathbf{y}^T \mathbf{x} = \pi' \tag{2}$$

We prove the existence of vector $\mathbf{x}$ by showing that

$$\mathbf{x} = \frac{\pi(P - P^-)}{\pi(P - P^-)\mathbf{e}^{\,T}} \tag{3}$$

[4] n is the number of states in the original chain P.

satisfies the requirements stated in the theorem i.e., when x is defined as in (3), $\pi' = c\pi$ (where c is the normalization factor $c = \pi' \mathrm{e}^{\,T}$).
Substituting the proposed expression for x in equation (2) we get:

$$\pi' P^- + \pi' \mathrm{y}^T \frac{\pi(P - P^-)}{\pi(P - P^-)\mathrm{e}^{\,T}} = \pi'$$

Now let's verify that $c\pi$ is a solution for this equation:

$$c\pi P^- + c\pi \mathrm{y}^T \frac{\pi(P - P^-)}{\pi(P - P^-)\mathrm{e}^{\,T}} = c\pi$$

by substituting y^T with its definition we get:

$$c\pi P^- + c\pi(P - P^-)\mathrm{e}^{\,T} \frac{\pi(P - P^-)}{\pi(P - P^-)\mathrm{e}^{\,T}} = c\pi$$

after some simplifications the equation is transformed into

$$\pi P = \pi$$

which holds true by hypothesis[5]. Hence when x is defined as in (3), $(c\pi, 1 - c\pi \mathrm{e}^{\,T})$ is indeed a solution for the derived system (1). □

At this point we can use Courtois and Semal's technique to compute bounds on the conditional probabilities of a subset of states in a system when only the transition probabilities among the states in the subset are known [3]. In this case the subset of states in the system represented by P_s is the set of original states, and the rest of the system is the extra state. Since the bounds can be computed without knowledge of the transition probabilities from the rest of the system to the selected subset of states there is no problem in not knowing the values of the elements of x. However if something is known about x then better bounds can be computed. Some information about x can be easily derived from P^ϵ, namely any column j of P^ϵ with all zero elements, corresponds to a zero element in x. Theorem 1 takes into account this factor by defining the set J of Z rows that contribute to the computation of bounds.

So far, the lumpability property of P^- has not been exploited. Observe that the MC P_s in general does not satisfy the lumpability conditions with respect to a given $\mathcal{A}$.

For example matrix P_s depicted below and derived from matrix P of Figure 1 is not lumpable.

$$P_s = \begin{array}{|c|c|cc|cc|c|c|}
\hline
 & 1 & 2 & 3 & 4 & 5 & 6 & \mathrm{s} \\
\hline
1 & \bullet & 2\lambda_1 & \lambda_2 & & & & \\
\hline
2 & & \bullet & & \lambda_1 & \lambda_1 & & \epsilon \\
3 & & & \bullet & & 2\lambda_1 & & \\
\hline
4 & & & & \bullet & & \lambda_1 & \epsilon \\
5 & & & & & \bullet & \lambda_1 & \\
\hline
6 & \mu & & & & & \bullet & \\
\hline
\mathrm{s} & \multicolumn{6}{c|}{\mathrm{x}} & 0 \\
\hline
\end{array}$$

[5]Constant c can be found using the normalization equation $c + a = 1$; furthermore the solution $\pi' = c\pi$ is unique since if P is irreducible, so is P_s.

But now let's modify the column $\mathbf{y}^T$ so that the lumpability conditions are satisfied. This can be done by increasing some of the elements of $\mathbf{y}^T$ and decreasing the matrix diagonal elements by the same quantity[6].

$$\tilde{P}_s = \begin{array}{|c|c|cc|cc|c|c|}
\hline
 & 1 & 2 & 3 & 4 & 5 & 6 & s \\
\hline
1 & \bullet & 2\lambda_1 & \lambda_2 & & & & \\
\hline
2 & & \bullet & & \lambda_1 & \lambda_1 & & \epsilon \\
3 & & & \bullet - \epsilon & & 2\lambda_1 & & \epsilon \\
\hline
4 & & & & \bullet & & \lambda_1 & \epsilon \\
5 & & & & & \bullet - \epsilon & \lambda_1 & \epsilon \\
\hline
6 & \mu & & & & & \bullet & \\
\hline
s & \multicolumn{6}{c|}{\mathbf{x}} & 0 \\
\hline
\end{array}
\qquad
\mathcal{P}_s = \begin{array}{|c|c|c|c|c|c|}
\hline
 & a1 & a2 & a3 & a4 & s \\
\hline
a1 & \bullet & 2\lambda_1 + \lambda_2 & & & \\
a2 & & \bullet & 2\lambda_1 & & \epsilon \\
a3 & & & \bullet & \lambda_1 & \epsilon \\
a4 & \mu & & & \bullet & \\
\hline
s & \multicolumn{4}{c|}{x} & 0 \\
\hline
\end{array}$$

The modified matrix $\tilde{P}_s$ is lumpable ($\mathcal{P}_s$ is the lumped chain) and the conditional probabilities of the aggregate states are still bounds for the sum of the probabilities in the aggregates computed on the original model. This statement is true because in the above proof we didn't make any assumption about P^- and P^ϵ other than $P^- \leq P$. In the sequel, n' is used to denote the number of aggregate states (i.e., $n' = |\mathcal{A}|$) and $\mathcal{S}$ is used to denote the first n' states of $\mathcal{P}_s$. Courtois and Semal's method can now be applied to the new lumped model as well.

Bounds on accumulated reward Often the performance indices of interest are not the steady state probabilities, but rather the accumulated reward given a reward function r from the set of states to $\mathbb{R}$. When the bounding method proposed has to be applied, it may be the case that aggregated states do not have the same associated reward. Of course if every state in aggregate ai had the same reward r_{ai}, then the aggregate reward would be r_{ai}, otherwise the computation of the exact reward associated with the aggregate would require the knowledge of the conditional probability distribution π^{ai} of the states in the aggregate:

$$r_{ai} = \sum_{s \in ai} \pi_s^{ai} r_s.$$

In this case the method described in Section 3.1 for the computation of bounds on the mean accumulated reward needs some minor adjustment. For each aggregate ai we have to compute the minimum and maximum reward rate:

$$r_{ai}^- = min_{s \in ai} r_s$$

$$r_{ai}^+ = max_{s \in ai} r_s$$

then instead of computing $\bar{R}_i$ we compute $\bar{R}_i^+$ using for each aggregate the maximum reward and $\bar{R}_i^-$ using for each aggregate the minimum reward. The lower and upper bounds for $\bar{R}$ are then derived as $min_i \bar{R}_i^-$ and $max_i \bar{R}_i^+$ respectively. Notice that this is the best we can do if we don't have any information about the conditional probability of the states in each aggregate, but this approach could result in loose bounds.

[6]Observe that the diagonal element could become negative by subtracting ϵ. To overcome this problem we can scale the diagonal of the matrix by performing the following transformation preserving the steady state probability result π

$$P \rightarrow \alpha P + (1 - \alpha) I$$

where $0 < \alpha \leq 1$.

Interpretation in terms of Bounded Aggregation The previous method can be interpreted in a different way, as a simplified version of the bounded aggregation technique [2]. However, the former interpretation is useful for devising bound improvement methods as we'll see in the next section.

The bounded aggregation method consists of two steps:

1. computation of bounds on the conditional state probabilities within each aggregate.
2. computation of bounds on the probability of being in each aggregate.

The bounds obtained in the two steps are then combined to obtain bounds on the steady state probability vector over the original set of states.

We are actually interested in computing only the probability vector of the aggregates (second step) since our aim is to deal directly with the lumped process. The $n' \times n'$ upper-leftmost submatrix $\mathcal{P}^-$ of the lumped matrix $\mathcal{P}_s$ derived in the previous paragraph is a lower bound for the correct inter-aggregates transition probability matrix $\mathcal{P}$ associated with P given state partition $\mathcal{A}$; indeed the correct transition probability between aggregate I and aggregate J is a convex combination of the rowsums in the submatrix $P_{I,J}$ (where the weight used for each row in the submatrix is the conditional probability of the corresponding state in aggregate I). In our approach we are giving a weight of one to the minimum rowsum, thus obtaining a lower bound with respect to the correct convex combination of the rowsums. In our example, matrix $P_{a2,a3}$ is

	4	5
2	λ_1	λ_2
3		$2\lambda_1$

and the transition probability between $a2$ and a_3 in $\mathcal{P}^-$ is $min(2\lambda_1, \lambda_1 + \lambda_2)$ while the corresponding value in $\mathcal{P}$ would be $2\alpha\lambda_1 + (1-\alpha)(\lambda_1 + \lambda_2)$ for some $0 \leq \alpha \leq 1$.

Observe that an upper bound, $\mathcal{P}^+$, for the probability between two aggregates could be computed as well by choosing the maximum among the rowsums in the submatrix (in the example above $max(2\lambda_1, \lambda_1 + \lambda_2)$ is an upper bound for the transition probability between $a2$ and $a3$).

3.3 Application of the method to the multiserver example

The model of Figure 2 is a good candidate for the application of the method. In fact this is a typical case in which the partition of the servers in two subclasses with equal speed within each subclass leads to a quasi-lumpable MC structure. We assume that the rates λ_1 and λ_2 associated with the two classes of servers are $\lambda_1 = 1.00$ and $\lambda_2 = 1.01$.

In Appendix A the complete state transition matrix of the model in Figure 2 is depicted; the states are grouped into *quasi-lumpable* sets. In Figure 6 the aggregated matrix (split into two parts due to space constraints) and the vector $\mathbf{y}^T$ are shown . All we know about $\mathbf{x}$ is that its 13th element must be zero (because the corresponding column is not modified to get $\mathcal{P}^-$ from P). In Figure 4 the description of the states for the aggregated model is given.

The state description for the complete model is not given here: it is the intermediate detail level representation $R2$ presented in Section 2. The reward rates for all aggregates used to

State #	Description (Clients in each server)	State #	Description (Clients in each server)
1	Serv1: 5, Serv2: 0	10	Serv1: 1. Serv2: 2 q of 2
2	Serv1: 4, Serv2: 1 q of 1	11	Serv1: 1, Serv2: 1 q of 1 & 1 q of 3
3	Serv1: 3, Serv2: 2 q of 1	12	Serv1: 1, Serv2: 1 q of 4
4	Serv1: 3, Serv2: 1 q of 2	13	Serv1: 0, Serv2: 3 q of 1 & 1 q of 2
5	Serv1: 2, Serv2: 3 q of 1	14	Serv1: 0, Serv2: 1 q of 1 & 2 q of 2
6	Serv1: 2, Serv2: 1 q of 1 & 1 q of 2	15	Serv1: 0, Serv2: 2 q of 1 & 1 q of 3
7	Serv1: 2, Serv2: 1 q of 3	16	Serv1: 0, Serv2: 1 q of 2 & 1 q of 3
8	Serv1: 1, Serv2: 4 q of 1	17	Serv1: 0, Serv2: 1 q of 1 & 1 q of 4
9	Serv1: 1, Serv2: 2 q of 1 & 1 q of 2	18	Serv1: 0, Serv2: 1 q of 5

Figure 4: States description

compute the mean queue length at the entrance of the multiple server ($S1 - S4$) and the throughput[7] out of $S0$ are shown in Figure 5. Observe that in this example we were able to compute an exact reward for each aggregate because all the aggregated states have the same associated reward.

Solving the 17 (aggregated) models (each with only one element of $\mathbf{x}$ equal to 1 and the others equal to 0) we obtain the following steady state probabilities, mean queue length, and throughput bounds. The table also contains the correct values computed on the complete model (that has 50 states).

[7] μ is the service rate of $S0$.

State #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	1 6	17	18
MQL	0	1	2	2	3	3	3	4	4	4	4	4	5	5	5	5	5	5
Throughput	μ	μ	μ	μ	μ	μ	μ	μ	μ	μ	μ	μ	0	0	0	0	0	0

Figure 5: Reward rates

	Lower bound	Exact value	Upper bound
Prob(State 1)	0.307430	0.321631	0.329239
Prob(State 2)	0.307430	0.320039	0.323022
Prob(State 3)	0.115106	0.119420	0.122238
Prob(State 4)	0.078126	0.079616	0.084354
Prob(State 5)	0.019179	0.019804	0.022648
Prob(State 6)	0.057889	0.059415	0.063442
Prob(State 7)	0.019131	0.019806	0.026669
Prob(State 8)	0.001198	0.001232	0.003351
Prob(State 9)	0.014373	0.014780	0.018140
Prob(State 10)	0.007139	0.007389	0.011374
Prob(State 11)	0.014219	0.014782	0.019938
Prob(State 12)	0.004703	0.004928	0.012848
Prob(State 13)	0.001194	0.001226	0.003879
Prob(State 14)	0.003562	0.003677	0.007472
Prob(State 15)	0.003544	0.003677	0.007506
Prob(State 16)	0.003527	0.003677	0.009229
Prob(State 17)	0.003506	0.003677	0.009698
Prob(State 18)	0.001164	0.001226	0.012440
Mean Queue Length	1.250705	1.273430	1.385868
Throughput	0.967612	0.98284	0.983502

The spread of the bounds for the mean queue length and throughput is 11% and 1.6% respectively.

The example model satisfies the constraints for the applicability of Theorem 3 that exploits the upper bound $\mathcal{P}^+$, so that we could use that theorem to obtain improved bounds. In Section 4 we show a technique for bounds improvement that doesn't require the computation of the inverse and the verification of constraints to be applied.

The reader may object that good bounds for the throughput and mean queue length of this specific example could be found by just observing that the (actually lumpable) model obtained by setting the service time of all the servers in the multiple server to the minimum among the service times in the original model, gives an upper bound for the throughput and a lower bound for the queue length. If the maximum among the service times in the original model is assigned to the servers, then we get a lower bound for the throughput and an upper bound for the queue length. This can be proved by induction on the number of customers in the system. Note however that it is not possible to compute bounds on the state probability distribution in the same simple way. Another possible objection is that the model is a product form queueing network, and efficient algorithms exist for the computation of exact results, however if we changed the policy for choosing the queue to join in the multiserver to a load dependent one (e.g., shortest-queue), product form solutions would not be applicable any more, while our method still applies. This example was chosen on purpose for its simplicity, in general it is not easy to prove that by lowering/increasing a rate in the system we get a bound on a given performance index. In Section 5 a more complex example is presented in which the presence of synchronization among the system components makes it difficult to find any intuitive argument for simple bounds computation.

3.4 Discussion on the computational advantages of the method

In this section we'll discuss the proposed method from the point of view of its convenience in terms of computational complexity. The main advantage of the method is that it allows us to deal with a smaller system than the one we start with, however there is a price we have to pay to be able to work with the reduced model. In fact there are at least two kinds of overhead we have to take into account in evaluating the method:

- derivation of the reduced matrix;
- computation of bounds.

Concerning the derivation of the reduced matrix, the ideal situation would be to derive the lumped matrices $\mathcal{P}^-$ and $\mathcal{P}^+$ directly from the high level model without having to compute (or at least to store) the large complete matrix. We'll show how this can be done when the high level formalism used is that of SWNs.

The computational complexity issue insteads, is related to the fact that the computation of bounds requires the solution of n' matrices of dimension $n' \times n'$ in the worst case. Later in this section we'll discuss how much smaller the lumped matrix must be for the method to be more efficient from the time complexity point of view, considering also the possibility of solution parallelization. The space complexity issue is discussed as well.

3.4.1 Construction of the lumped matrix

The issue of automatic construction of the lumped matrix is closely related to that of the detection of the symmetries on the state space that induce a partition into state aggregates. The same problem has to be faced when exact lumping methods are used, indeed our bounding methods can be considered as an extension of the existing exact lumping methods.

Observe that usually symmetries are present in the state space whenever the modeled system has a symmetric structure. For example a system composed of a number of similar objects that behave homogeneously usually has a symmetric state space. Although this observation is rather intuitive, in general is not that obvious how to define the equivalence relation among states that allow to group them into equivalence classes (i.e., into aggregates).

In [1] it has been shown that when a system is described using the modeling formalism of Stochastic Well-Formed Colored Petri Nets (SWNs), most of the symmetries of the state space can be automatically discovered and exploited. A Symbolic Reachability Graph (SRG) construction algorithm has been defined that directly builds the lumped state space and corresponding CTMC. From the lumped CTMC it is possible to derive both high-level and detailed performance indices. We now show that same approach can be extended to our bounding method, that is the upper and lower bound aggregated matrices $\mathcal{P}^-$ and $\mathcal{P}^+$ can be automatically built using a modified version of the *SRG* algorithm. The bounding matrices can be computed at different accuracy levels; as usual higher accuracy can be achieved with higher computational cost.

Automatic construction of a lumped matrix from a SWN model In Section 2 we introduced the concept of color classes and subclasses in SWN. The color classes are used to define sets of "similar" objects and the partition of a class into subclasses is used to identify subsets of objects in a class that share the same behavior. As already pointed out, it is possible to distinguish between two possible situations: (1) the objects in different subclasses have different *qualitative* behavior, i.e., they cannot play the same "role" in the system because they have different possible evolutions, (2) the objects in different subclasses have the same *qualitative* behavior, but the event sequences happen with different rates depending on which subclass the object belongs to.
For example, in the model of Figure ?? the colored tokens representing customers in the "multiserver" station follow the same route through transition *end_serv*2, independently of their color, however, the firing rate of the transition depends on the token color.

Hence, given a SWN model of the system under study, a first analysis is needed to detect all the subclasses in each class with similar "qualitative" behavior. This permits automatic determination of the candidate quasi-lumpable state aggregates. Then two approaches are possible, (1) apply the Symbolic Reachability Graph generation algorithm to the system as it is specified and use the information about aggregate states to compute the lumped matrices $\mathcal{P}^-$ and $\mathcal{P}^+$ in a second step; (2) apply the SRG generation algorithm to a modified system with some of the homogeneously behaving subclasses merged in such a way that $\mathcal{P}^-$ and $\mathcal{P}^+$ are directly derived.

The two algorithms that follow are respectively the subclasses merge algorithm and the modified model construction algorithm, where the modified model is the one from which the aggregate matrices $\mathcal{P}^-$ and $\mathcal{P}^+$ can be directly derived.

Subclasses merge algorithm For each basic color class, the following static classes merge procedure has to be performed:

```
$old\_static_i$ $=$ ⟨set of static subclasses in the original $C_i$⟩
$new\_static_i$ $= \emptyset$
while $old\_static_i \neq \emptyset$ do
    ⟨ remove a subclass $sc$ form $old\_static_i$⟩
    $lsc = sc$
    $sc\_list = emptylist$
    append($sc, sc\_list$)
    for each $sc' \in old\_static_i$ do
        if $semilumpable(sc, sc')$ then
            $lsc = lsc \cup sc'$
            ⟨ remove $sc'$ form $old\_static_i$⟩
            append($sc', sc\_list$)
            end if
        end for
        $old\_sc\_list[lsc] = sc\_list$
        $\langle$ add static subclass $lsc$ to $new\_static_i \rangle$
end while
```

A set of static subclasses *old_static*$_i$ is the input for this procedure. The output is a new set *new_static*$_i$, of static subclasses. The cardinality of the new set is less than or equal to that of the old set. For each new static subclass *lsc*, a list *old_sc_list*[*lsc*] of the old static subclasses that have been merged into *lsc* is maintained. Function *semilumpable*(sc, sc') returns *true* iff for all transitions that have C_i in their color domain definition, no predicates of type $d(X_i^j) = sc$, $d(X_i^j) = sc'$, $d(X_i^j) = d(X_i^k)$ or their negation are used either in the transition predicate or in the function predicates labeling the arcs connected to the transition (i.e., neither the transition enabling conditions nor the transition state change can depend on the fact that some element in the transition color instance belong or not to sc or sc'). Observe that more complex but less restrictive conditions may be defined: we might for example allow the presence of predicate $d(X_i^j) = sc$ provided that it is in *or* with $d(X_i^j) = sc'$.

Aggregate model computation algorithm The aggregate matrices $\mathcal{P}^-$ and $\mathcal{P}^+$ are computed by applying the usual SRG generation algorithm to a modified model. The modified model has the same structure as the original one, while it differs from it in the static subclasses definition and, as a consequence, in the transition firing rates.

In a SWN, the transition firing rate of a generic transition t is defined as a function θ_t from tuples of static color classes to reals.

For example transition *compute* in Figure 7 takes a token with its two component color from its input place: the first component represents a processor and the second represents a job. If the jobs class J is divided into two subclasses of short jobs J_s and long jobs J_l and the processors are divided into two subclasses of fast processors P_f and slow processors P_s, then a possible definition for the transition rate could be: $\theta_t(\langle < P_s, J_s > \rangle) = 0.3$, $\theta_t(\langle < P_f, J_s > \rangle) = 1$, $\theta_t(\langle < P_s, J_l > \rangle) = 0.15$, $\theta_t(\langle < P_f, J_l > \rangle) = 0.5$.

When a transition instance $t(z)$ is fired, its rate $\lambda(t(z))$ is computed as follows: (1) derive from z the tuple d of corresponding static subclasses (each dynamic subclass is associated with exactly one static subclass, see [1]; (2) $\lambda(t(z)) = m\ \theta_t(d)$ where m is a factor that depends on the cardinality of both the subclasses in z and in d. The reason for the multiplicative factor is that a symbolic firing instance is an aggregation of m regular firing instances all with the same rate $\theta_t(d)$. In our example a possible symbolic instance for transition *Compute* could be $Compute(Z_P^1, Z_J^2)$ with $|Z_P^1| = 2, Z_P^1 \in P_s$, and $|Z_J^2| = 1, Z_J^2 \in J_l$ meaning that there are 2 $(= |Z_P^1|)$ slow processors each of which is processing 1 $(= |Z_J^2|)$ long job. This symbolic instance stands for 2 "ordinary" instances that are grouped in the lumping process.

In the following we show how to derive the modified model to compute the aggregate matrices given a specific aggregation of the static subclasses. We denote $D_{i,j}$ the new static subclasses and $\{D^l_{i,j}\}$ the set of the original subclasses that are aggregated into $D_{i,j}$.

The rates of the resulting new transitions, in general, will depend on the cardinality of the dynamic subclasses in the instantiation.

Let $t(z)$ be the transition instance for which a rate has to be computed. Let d be the associated static subclasses tuple. Let d' be the subtuple of d composed of aggregate static subclasses and z' the corresponding subtuple of z.

For each element $d_k = D_{i,j}$ in d', compute the set $\mathcal{Z}_k$ of possible partitions of dynamic subclass z_k into one or more new dynamic subclasses, each associated with a different static subclass $D^l_{i,j}$ of $D_{i,j}$ (see Figure 8).
The cartesian product of the $\mathcal{Z}_k$s leads to a set of sums of original transition instances. From each sum a rate can be computed that correspond to a value for a rowsum in the aggregate transition represented by $t(z)$. The minimum and maximum in this set of rates gives the value for the corresponding transition rate in the aggregate matrices.

In the example of Figure 7 assume that $|P_s| = |P_f| = 2$ and $|J_s| = 2, |J_l| = 1$, and suppose we want to merge slow and fast processors into a unique class P and short and long jobs into a unique class J. Hence the symbolic transition instance $Compute(Z^1_P, Z^2_J)$ with $|Z^1_P| = 2, Z^1_P \in P$ and $|Z^2_J| = 2, Z^2_J \in J$ incorporates the following 6 possible instances with respect to the old classes partition.

- $Compute(Z^{1a}_P, Z^{1a}_J) + Compute(Z^{1a}_P, Z^{1b}_J)$ with $|Z^{1a}_P| = 2$ and $Z^{1a}_P \in P_s$, $|Z^{1a}_J| = 1$ and $Z^{1a}_J \in J_s$, $|Z^{1b}_J| = 1$ and $Z^{1b}_J \in J_l$; corresponding rate 0.9
- $Compute(Z^{1a}_P, Z^{1a}_J) + Compute(Z^{1a}_P, Z^{1b}_J)$ with $|Z^{1a}_P| = 2$ and $Z^{1a}_P \in P_f$, $|Z^{1a}_J| = 1$ and $Z^{1a}_J \in J_s$, $|Z^{1b}_J| = 1$ and $Z^{1b}_J \in J_l$; corresponding rate 3
- $Compute(Z^{1a}_P, Z^{1a}_J)$ with $|Z^{1a}_P| = 2$ and $Z^{1a}_P \in P_s$, $|Z^{1a}_J| = 2$ and $Z^{1a}_J \in J_s$; corresponding rate 1.2
- $Compute(Z^{1a}_P, Z^{1a}_J)$ with $|Z^{1a}_P| = 2$ and $Z^{1a}_P \in P_f$, $|Z^{1a}_J| = 1$ and $Z^{1a}_J \in J_s$; corresponding rate 4
- $Compute(Z^{1a}_P, Z^{1a}_J) + Compute(Z^{1a}_P, Z^{1b}_J) + Compute(Z^{1b}_P, Z^{1a}_J) + Compute(Z^{1b}_P, Z^{1b}_J)$ with $|Z^{1a}_P| = 1$ and $Z^{1a}_P \in P_s$, $|Z^{1b}_P| = 1$ and $Z^{1a}_P \in P_f$, $|Z^{1a}_J| = 1$ and $Z^{1a}_J \in J_s$, $|Z^{1b}_J| = 1$ and $Z^{1b}_J \in J_l$; corresponding rate 3.3
- $Compute(Z^{1a}_P, Z^{1a}_J) + Compute(Z^{1b}_P, Z^{1a}_J)$ with $|Z^{1a}_P| = 1$ and $Z^{1a}_P \in P_s$, $|Z^{1b}_P| = 1$ and $Z^{1a}_P \in P_f$, $|Z^{1a}_J| = 2$ and $Z^{1b}_J \in J_s$; corresponding rate 2.6

There is a computationally less expensive but not always accurate method for computing lower/upper bounds for the elements of $\mathcal{P}^-$ and $\mathcal{P}^+$. Given a transition t and a tuple d of aggregate static subclasses, compute the cartesian product D of the sets of original subclasses in the component aggregate subclasses. Associate with d the $min(max)_{d' \in D}\theta_t(d')$. In this way the computed $\mathcal{P}^-$ and $\mathcal{P}^+$ are bounds for $\mathcal{P}$, but they can be very inaccurate in some cases. In the previous example this simplified method would have predicted correctly the maximum rate (= 4), while the minimum (= 0.6) would have been less than the the correct one (0.9).

Observe that even if the computation of the aggregate transition rates has a certain cost, it could be performed once and for all in a way that the result is easily reusable for models differing only in the actual parameter values.

3.4.2 Complexity of bounds computation

The main goal of the proposed method is to cope with the problem of state space explosion by allowing some aggregation of the state space. However since the aggregate chain is only a bound for the actual aggregate chain and since the computation of bounds of a chain given its lower (and/or upper) bound transition probability matrix is more expensive than just computing the steady state solution of a chain of the same dimension, in some cases the method proposed could be computationally more expensive than the solution of the whole model we started with.

Let's consider the queueing network example we have been using throughout the paper. The complete model has 50 states while the aggregate model has 18 states. If we assume that the computation of the steady state solution of a MC of dimension n is $O(n^2)$ (we are assuming that sparse matrix algorithms can be applied, if this is not the case, the worst case complexity would be $O(n^3)$) then the time required for solving the complete model is proportional to $50^2 = 2500$, while the time required for getting bounds from the aggregate model is proportional to $17 * (18^2) = 5508$. Apparently the bounding method for this case is not computationally convenient, however the situation changes completely when the dimension of the studied system grows. The size of this model was chosen small to be able to show both the complete and aggregate transition matrices. Similar models with different number of customers, servers in the multiple server station and static subclasses in the class of servers, show how convenient the method can be as the size of the problem increases. This is shown in the table of Figure 9 where the number of states in the lumped (column a) and complete (column b) Markov chain and the corresponding estimated computational time cost as function of the number of servers and customers are listed. Notice that the number of states in the complete model refers to the case in which each queue in the multiserver has a different speed.

In general the complexity for computing bounds is $k(n')^2$ where k is the number of aggregate columns modified in the original MC to make it lumpable, hence in the worst case $k = n'$. In order for the method to be computationally convenient the ratio n/n' must be such that $k(n')^2 < n^2$.

Finally observe that even when the ratio between the time complexity of exact solution computation versus that of bounds computation is less than one, the computation of bounds can still be convenient. In fact the computation of bounds on the lumped model can be easily parallelized because it requires the solution of several smaller and completely independent (lumped) models, thus resulting in a gain in time complexity. Breaking up the problem into smaller ones may be also crucial from the space complexity point of view since the whole model could not fit into the available main memory, preventing the solution if no virtual memory is provided or causing a loss of performance due to frequent paging if virtual memory is available.

4 Improving bounds

The bounds found with the method described above may have a different degree of tightness depending on the sensitivity of the solution to the transitions replaced by lower bounds. In this section a technique is presented that can be used to improve the bounds when those obtained by directly applying the method presented in Section 3 are too loose. In the previous section we mentioned that it is possible to derive an upper bound $\mathcal{P}^+$ for the transition probabilities between aggregates; once we have this upper bound it is possible to use Theorem 3 to compute another set of bounds that, combined with those obtained from the lower bound $\mathcal{P}^-$, may lead to tighter bounds. The problem is that in this case the inverse $(I - P^+)^{-1}$ is not guaranteed to exist, and even if it does exist some further conditions must be verified in order to be sure that the results obtained are indeed bounds. In the next paragraph we will see how the upper bound on the aggregate transition probability can be used in conjunction with the the steady state probability bounds π^-, π^+ previously computed to get better bounds on any Markov reward function (and hence also on the state probabilities).

Exploiting information about the "probability redistribution" vector x The following method is aimed at improving the bounds on the performance measures we are interested in; we assume that each performance metric is defined by associating a reward function r on states with the model. In particular a possible performance metric could be the steady state probability of one specific state s_j; in this case all the states are assigned a null reward except s_j which is assigned a reward $r_j = 1$.

If the exact vector x of "redistribution" probabilities was known and substituted into matrix $\mathcal{P}_s$, then the mean cumulative reward (conditioned on being in the first n' states) computed from $\mathcal{P}_s$ would be the correct value we are looking for.

The behavior of the Markov chain described by $\mathcal{P}_s$ is the following: the evolution starts from one state in $\mathcal{S}$ and goes on until it reaches the extra state s, then the evolution starts again from a state in $\mathcal{S}$ probabilistically chosen according to the probability distribution defined by x until it reaches s again. The accumulated reward between two successive visits to s depends on which state is chosen to start with when exiting s. Assume we have the list $s_{j1}, s_{j2}, \ldots, s_{jn'}$ of states ordered in such a way that the accumulated reward before exiting to s, given that we started in $s_{jm}, m = 1, \ldots, n'$, is greater than or equal to the accumulated reward before exiting to s, given that we started in $s_{jk}, k > m$.

An upper bound on the mean cumulative reward can be computed from model $\mathcal{P}_s$ by substituting for x a modified vector x$'$ such that $\mathrm{x}'_{jm} \geq \mathrm{x}_{jm}$ $(1 \leq m \leq k \leq n')$ and $\mathrm{x}'_{jl} \leq \mathrm{x}_{jl}$ $(k < l \leq n')$.

A lower bound on the mean cumulative reward can be computed if in model $\mathcal{P}_s$ we substitute for x a modified vector x$'$ such that $\mathrm{x}'_{jm} \leq \mathrm{x}_{jm}$ $(1 \leq m \leq k \leq n')$ and $\mathrm{x}'_{jl} \geq \mathrm{x}_{jl}$ $(k < l \leq n')$.

If we were able to estimate an upper bound for x this could thus be used to compute better bounds on the desired performance figures. Recall that the correct x is defined as follows:

$$\mathbf{x} = \frac{\pi(\mathcal{P} - \mathcal{P}^-)}{\pi(\mathcal{P} - \mathcal{P}^-)\mathbf{e}^{\,T}}$$

Hence the following vector $\mathbf{x}^+$ is an upper bound for x:

$$\mathbf{x}^+ = \frac{\pi^+(\mathcal{P}^+ - \mathcal{P}^-)}{\pi^-(\mathbf{e}^{\,T} - \mathcal{P}^-\mathbf{e}^{\,T})}$$

where π^+ and π^- are upper and lower bounds on the steady state probability distribution (e.g., that were computed in a first step using the method presented in the previous section) and $\mathcal{P}^+$ is an upper bound on the inter-aggregate transition probability matrix.

Applying the above method to the queueing network example we obtain for the vector x the upper bound estimate in Figure 10.

The new (much improved) bounds computed for the mean queue length and throughput using this method are:

	Lower bound	Exact value	Upper bound	Spread %
Mean Queue Length	1.250705	1.273430	1.295075	3.5%
Throughput	0.980303	0.98284	0.983502	0.3%

5 An Application Example

In this section we present a more complex example in which the presence of synchronization constraints does not allow us to easily obtain bounds on the performance figures of interest using some other method.

The modeled system consists of a parallel program organized according to a master-slave computation paradigm. The program is mapped on a parallel architecture: we assume there is a processor for each process, that the processors are homogeneous, and that due to the type of the interconnection network the communication time between the master and the slaves is not homogeneous (some slaves are "closer" to the master than others). This is the case for the example of a master-slave program mapped onto a mesh architecture depicted in Figure 11. The processors are represented by squares, while the processes are represented by circles. In the figure both the physical channels (connecting the processors) and the logical channels (connecting the processes) are depicted.

The master divides the problem to be solved into several subproblems, and as long as there are free slaves it delivers the tasks to be executed to them. When no more free slaves are available, the master waits for some slave to complete its current task. The master also performs some computation in parallel with communicating, to process the results received to prepare new data to be distributed.

We have modeled the master-slave system using the SWN formalism; in Figure 12 the resulting model is depicted. We do not include the definition of SWNs here due to space constraints, however since the SWN model of the program resembles a flow-chart, we can explain its semantics intuitively. The rectangles are called *transitions* and stand for statements in a process code, the circles are called *places*, and may contain *tokens*. In this model the presence of a token in one place indicates the program counter position for the corresponding process, so that the next statement to be executed is represented by the transition pointed by the arc exiting the marked place.

The model is composed of two subnets, the first (leftmost) one represents the master's behavior while the second (rightmost) one represent the common behavior of the slaves. The distinction among the three slaves is obtained by using *distinguishable* tokens, i.e., tokens in the slave behavior subnet are labeled with a slave identifier. The behavior model has been simplified as much as possible to make the picture readable. The master is composed of two parts that work in parallel: the computation part, consisting of a certain number of iterations of two procedures (transitions *inLoop*, *cmpMstA*, *cmpMstB*), and the synchronous communication part consisting of a certain number of iterations of send/receive operations (transitions *snd*, *rx1*, *rx2* plus the transitions representing actual communication, shared with the slaves net: *begRx* , *endRx* , *begSnd*, *endSnd*). The slaves behavior can be described as repeated iteration of three operations: receive (transitions *begRx* and *endRx*), compute (transition *compSL*), send (transitions *begSnd* and *endSnd*). There is only one color class needed: the slaves class S. It has cardinality three and is divided into two static subclasses $S1$ of cardinality 2 and $S2$ of cardinality 1. Subclass $S1$ represents the slaves that are closer to the master while subclass $S2$ represents the farther slave. As a consequence the transitions representing communication between the master and the slaves have a rate that depends on the slave identity.

The detailed state representation for this model is a list of program counter values, one for each process in the program. The MC representing this model behavior is quasi-lumpable with respect to the aggregation of states induced by considering the slaves as undistinguishable. In other words we aggregate all the states that are equal up to a permutation of slaves identities. For example we aggregate the state M_1 with slave $S1$ waiting to receive some data from the master, $S2$ sending a result to the master and $S3$ performing a computation, and state M_2 with slave $S2$ waiting to receive some data from the master, $S3$ sending a result to the master and $S1$ performing a computation. Observe that although this aggregation reflects a symmetry in the state space structure due to the homogeneous behavior of the slaves, there is a perturbation in the transition rates, due to the fact that the communication between $S3$ and the master takes longer than the communication between the other two slaves and the master.

The quasi-lumpable model has 109 states and the number of aggregate states is 49.

The performance measures we have considered are three:

- throughput of the system, i.e. the number of completed computations in the time unit (throughput of transition PAR representing the beginning of a new computation); acronym used in the table: THRUPUT;
- mean number of slaves waiting to receive a task from the master (i.e., mean number of tokens in place *Slaves*); acronym used in the table: MNSWRx;
- mean number of slaves waiting to send the result of their computation to the master

(mean number of tokens in place *endcmpSL*); acronym used in the table: MNSWSnd.

We have done three experiments changing the values of the difference between the rates associated with the communication between master and slaves (represented by transitions *endRx* and *endSnd*). We have fixed $\theta(endRx, S3) = 0.5$ and $\theta(endSnd, S3) = 3.5$ while the rates for $(endRx, S1-2)$ and $(endSnd, S1-2)$ that have been used in the three experiments are the following:

1. $\theta(endRx, S1) = 0.51$, $\theta(endSnd, S1) = 3.57$
2. $\theta(endRx, S1) = 0.525$, $\theta(endSnd, S1) = 3.675$
3. $\theta(endRx, S1) = 0.55$, $\theta(endSnd, S1) = 3.85$

Experiment #	Perf. Measure	Lower bound	Upper bound	Spread %
1	THRUPUT	0.012396	0.014595	18%
	MNSWSnd	0.079165	0.137257	73%
	MNSWRx	2.547911	2.726692	7%
2	THRUPUT	0.011936	0.016718	40%
	MNSWSnd	0.056813	0.180464	218%
	MNSWRx	2.436226	2.802309	15%
3	THRUPUT	0.011333	0.019553	73%
	MNSWSnd	0.039295	0.257986	557%
	MNSWRx	2.249367	2.861513	27%

The results shown in the table above are obtained from the direct method. We have also applied the bounds improvement method: it resulted in tightened bounds in the first experiment but it did not give any substantial improvement in experiments 2 and 3. As a matter of fact, the first case is also the only one that is eligible for the application of Theorem 3 of Section 3.

Clearly there is a strict correlation between the difference $P^+ - P^-$ and the spread in the bounds. It might be worthwhile to assess a priori the sensitivity of the system on variations on the transition probabilities that are positive in $P^+ - P^-$.

In this example we have also observed that some further bounds refinement could be obtained by the consideration of the high level model. If no contention for resources were present in the slaves subnet, then we would know for sure that the throughput of slave $S3$ is lower than the throughput of slaves $S1$ and $S2$. Since the slaves compete to communicate with the master, the throughput of both kind of slaves decreases with respect to the no contention case; moreover the difference between the throughput of the slave $S3$ and the other two slaves becomes smaller as the probability of contention increases. Our conjecture is that in the system we are studying, the throughput of slave $S3$ is less than or equal to the throughput of the other two slaves. Assuming that the conjecture is true (and it is indeed true for our experiment cases), we show how this information can be used to improve the estimate of the lower bound matrix $\mathcal{P}^-$. Of course, the kind of property to be proved about the high level model in order to obtain improved estimates of $\mathcal{P}^-$ and/or $\mathcal{P}^+$ and the technique used to prove it depend heavily on the kind of high level formalism adopted and in general cannot be automated.

Notice that in our example, every aggregate transition submatrix $P_{I,J}$ with different row-sums, refers to a communication transition (let's call this transition t_{comm}). Since only one slave at a time can communicate, each quasi-lumpable communication transition aggregate contains two kinds of ordinary transitions: one for the slave $S3$ and the other one for the slaves $S1$ and $S2$. Thus a lower bound for $\mathcal{P}_{I,J}$ is given by:
Pr^{+}(comm. by the slave in $S2$) $\theta(t_{comm}, S2)$ + Pr^{-}(comm. by a slave in $S1$) $\theta(t_{comm}, S1)$ with the constraint Pr^{+}(comm. by the slave in $S2$) + Pr^{-}(comm. by a slave in $S1$) = 1.

The probability p that place $inREC$ contains a token representing slave $S3$ given that $inREC$ is not empty can be formulated as

$$p = \frac{\frac{throughput(endRx,S2)}{\theta(endRx,S2)}}{\frac{throughput(endRx,S1)}{\theta(endRx,S1)} + \frac{throughput(endRx,S2)}{\theta(endRx,S2)}}$$

An upper bound for this probability can be found by considering the throughput of the slave in $S2$ to be equal to the throughput of both slaves in $S1$ (let's call the common throughput value t):

$$p^{+} = \frac{\frac{t}{\theta(endRx,S2)}}{\frac{2\,t}{\theta(endRx,S1)} + \frac{t}{\theta(endRx,S2)}}$$

and after a simplification

$$p^{+} = \frac{\theta(endRx, S1)}{2\;\theta(endRx, S2)\; + \;\theta(endRx, S1)}$$

A similar argument applies to place $inSnd$.

Thus a lower bound for the rate of an aggregate transition representing a communication is obtained by weighing by p^{+} the row-sum corresponding to a communication transition involving the slave in $S2$, and weighing by $1 - p^{+}$ the remaining communication transitions associated with the slaves in $S1$.

The results obtained by applying this technique to the model of experiment 3 are the following:

Perf. Measure	Lower bound	Upper bound	Spread %
THRUPUT	0.012154	0.015649	29%
MNSWSnd	0.069087	0.161423	134%
MNSWRx	2.507557	2.773172	11%

The bounds are definitely tighter than those obtained in experiment three ignoring the model semantics. This indicates that whenever some additional information about the system behavior can be derived from the high level model, it should be exploited to get improved bounds.

6 Conclusions

In this paper a new method for the computation of bounds on the steady state probability of quasi-lumpable Markov chains is described. The contribution of the paper is twofold. In the context of solutions of large Markov chains can be defined as an extension of lumpability methods to quasi-lumpable models. In the paper we show how a quasi-lumpable MC can be transformed into a lumpable one and how bounds on the desired performance indices can be obtained from the lumped modified matrix using the Courtois and Semal results of [2, 3].

In the framework of analysis methods for Stochastic Well-Formed Nets [1], we have generalized the Symbolic Reachability Graph analysis technique: in the paper we have shown that the modified lumped matrix needed to compute bounds can be directly computed from the SWN description of a class of quasi-lumpable models.

A class of models that are good candidates for the application of the method is the one representing systems that are composed of several homogeneously behaving objects (like the slaves in the examples) that can only differ slightly in the timing of some operation (the communication with the master in the example). The matrices $\mathcal{P}^-$ and $\mathcal{P}^+$ can be directly computed for models in this class represented with the formalism of SWN.

Another class of models that can be analyzed with the proposed bounding method is that representing systems were the departure/arrival rates are state dependent and are a smooth function of the system population. Some experiments on models in this class are being performed.

We have also devised a method to improve the bounds when both upper and lower bounds ($\mathcal{P}^+$ and $\mathcal{P}^-$) for the aggregate transition probability matrix ($\mathcal{P}$) are available. With respect to Courtois and Semal's results, this method could be used to obtain better bounds when the inverse $(I - P^+)^{-1}$ doesn't satisfy the constraints required to apply Theorem 3.

Finally we have applied the method to several example models. Future developments of this work will be in the direction of (1) devising new techniques to further improve the bounds possibly by using some information from the high level model and (2) studying the application of the method to some interesting cases where particular kinds of aggregation criteria apply.

References

[1] G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad. Stochastic well-formed coloured nets and multiprocessor modelling applications. In K. Jensen and G. Rozenberg, editors, *High-Level Petri Nets. Theory and Application.* Springer Verlag, 1991.

[2] P.J. Courtois and P. Semal. Bounds for positive eigenvectors of nonnegative matrices and their approximations. *Journal of the ACM*, 31(4):804–825, Oct. 1984.

[3] P.J. Courtois and P. Semal. Computable bounds on conditional steady-state probabilities in large markov chains and queueing models. *IEEE Journal on Selected Areas in Communication*, SAC-4(6):926–937, Sept. 1986.

[4] K. Jensen. Coloured Petri nets. In W. Brawer, W. Reisig, and G. Rozenberg, editors, *Advances on Petri Nets '86 - Part I*, volume 254 of *LNCS*, pages 248–299. Springer Verlag, Bad Honnef, West Germany, February 1987.

[5] J.G. Kemeni and J.L. Snell. *Finite Markov Chains.* Van Nostrand, Princeton, NJ, 1960.

[6] V. F. Nicola. Lumping in markov reward processes. In *Proc. 1st International Conference on Numerical Solution of Markov Chains*. Marcel Dekker, Inc., January 1990.

[7] W. H. Sanders and J. F. Meyer. Reduced base model construction methods for stochastic activity networks. In *Proc. International Conference on Petri Nets and Performance Models*, pages 74–84, Kyoto,Japan, December 1989.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	y
1		1.000																	0
2	1.000		0.750	0.250															0.01
3		2.000			0.500	0.500													0.02
4		1.000				0.750	0.250												0.01
5			3.000					0.250	0.750										0.02
6			1.000	1.000					0.500	0.250	0.250								0.02
7				1.000							0.750	0.250							0.02
8					4.020								1.000						0.00
9					1.000	2.000							0.250	0.500	0.250				0.02
10						2.000								0.500		0.500			0.02
11						1.000	1.000								0.500	0.250	0.250		0.02
12							1.000										0.750	0.250	0.02
13								1.000	3.000										0.02
14									2.000	1.000									0.02
15									1.000		2.000								0.02
16										1.000	1.000								0.02
17											1.000	1.000							0.02
18												1.000							0.01

Figure 6: The aggregated matrix $\mathcal{P}$ and the corresponding vector $\mathbf{y}^T$.

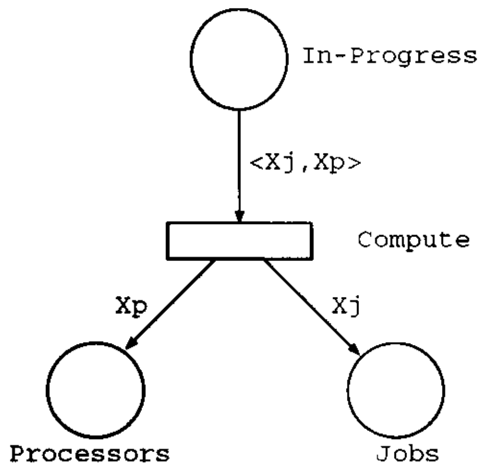

Figure 7: A simple SWN transition example.

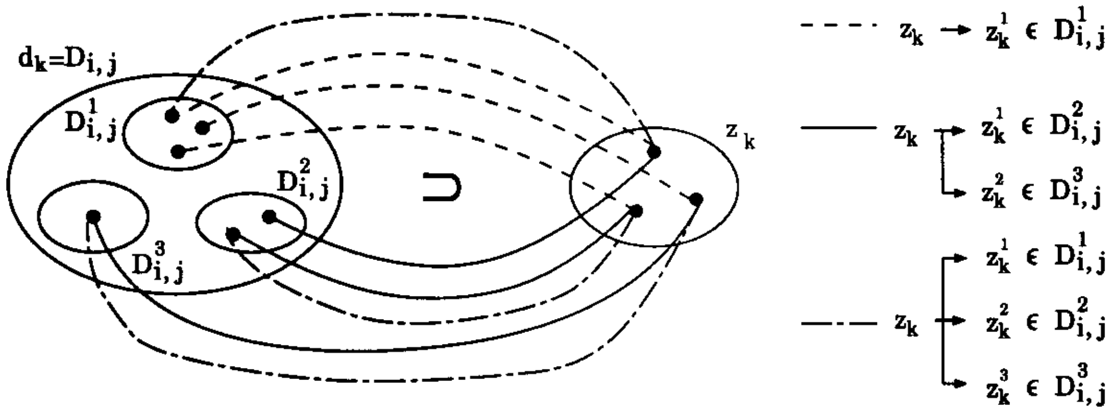

Figure 8: Possible partitions of z_k's with respect to the "merged" static subclass $D_{i,j}$.

# Serv.	# Aggr. States	# Non Aggr. States	(a) Bounds Comp. Time	(b) Exact Sol. Comp. Time	(b)/(a)
5 customers					
4	18	126	5832	15876	2.72
5	19	252	6859	63504	9.26
6	19	462	6859	213444	31.12
8 customers					
4	53	495	148877	245025	1.64
5	60	1287	216000	1656369	7.6
6	64	1716	262144	2944656	11.23
10 customers					
4	94	1001	830584	1002001	1.21
5	113	3003	1442897	9018009	6.25
6	125	8008	1953125	64128064	32.83

Figure 9: Comparing time complexity for the computation of bounds vs. exact solution.

State #	1	2	3	4	5	6	7	8	9
x^+	0.350781	0.840613	0.609799	0.258352	0.07127	0.434729	0.08 62158	0.0042123	0.100793
State #	10	11	12	13	14	15	16	17	18
x^+	0.0733268	0.160317	0.048524	0.000000	0.0162282	0.0163021	0.0400884	0.0421257	0.0135091

Figure 10: Upper bound for vector x.

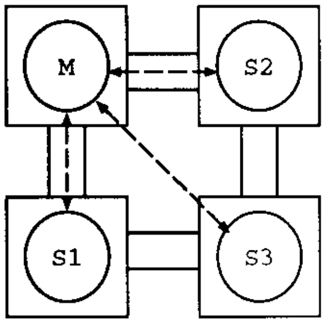

Figure 11: A Parallel Program Model.

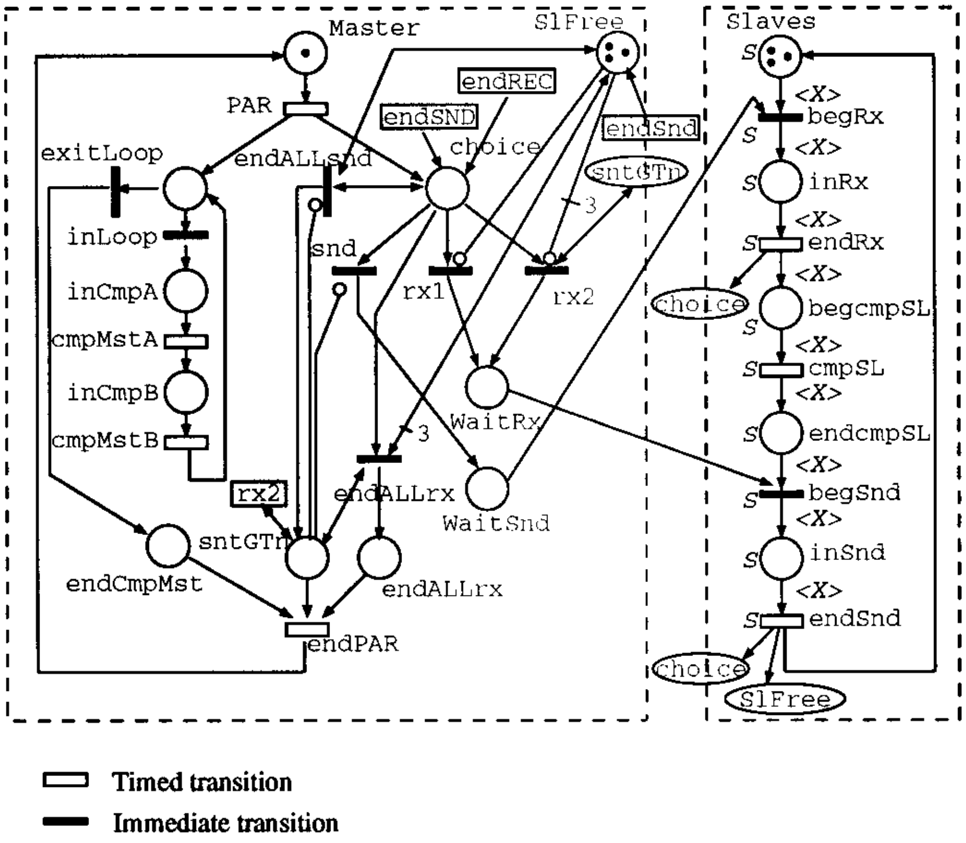

Figure 12: The SWN model of the Master-Slave Program.

Appendix

A Complete matrix for the multiserver example

	1	2	3	4	6	7	5	8	10	13	9	12	14	15
1		0.500	0.500											
2	1.000			0.250	0.500		0.250							
3	1.010				0.500	0.250		0.250						
4		2.000							0.500		0.500			
6		1.010	1.000						0.250	0.250		0.250	0.250	
7			2.020							0.500				0.500
5		1.000									0.250	0.500		
8			1.010										0.500	0.250
10				1.010	2.000									
13					2.020	1.000								
9				1.000			1.000							
12					1.000		1.010							
14					1.010			1.000						
15						1.010		1.010						
11							1.000							
16								1.010						
20									2.020	2.000				
19									1.000		1.010	1.000		
21									1.010				2.000	
24										1.000		2.020		
26										1.010			1.010	1.000
17											2.000			
25												1.010	1.000	
28														2.020
18											1.000			
23												1.000		
27													1.010	
29														1.010
22														
30														
35														
37														
32														
36														
43														
45														
34														
38														
41														
46														
31														
42														
44														
48														
33														
40														
47														
49														
39														
50														

	11	16	20	19	21	24	26	17	25	28	18	23	27	29
1														
2														
3														
4														
6														
7														
5	0.250													
8		0.250												
10			0.250	0.500	0.250									
13			0.250			0.250	0.500							
9				0.500				0.250			0.250			
12				0.250		0.250			0.250			0.250		
14					0.250		0.250		0.250				0.250	
15							0.500			0.250				0.250
11											0.250	0.500		
16													0.500	0.250
20														
19														
21														
24														
26														
17														
25														
28														
18	1.000													
23	1.010													
27		1.000												
29		1.010												
22	1.000													
30		1.010												
35			1.000	2.020		1.000								
37			1.010		1.010		2.000							
32				2.000				1.010						
36				1.010	1.000				1.000					
43						1.010	1.000		1.010					
45							2.020			1.000				
34				1.000							1.010	1.000		
38					1.010								2.000	
41						1.000						2.020		
46							1.010						1.010	1.000
31								1.000			1.000			
42									1.000			1.010		
44									1.010				1.000	
48										1.010				1.010
33											1.000			
40												1.000		
47													1.010	
49														1.010
39														
50														

	22	30	35	37	32	36	43	45	34	38	41	46
1												
2												
3												
4												
6												
7												
5												
8												
10												
13												
9												
12												
14												
15												
11	0.250											
16		0.250										
20			0.500	0.500								
19			0.250		0.250	0.250			0.250			
21				0.250		0.500				0.250		
24			0.250				0.500				0.250	
26				0.250			0.250	0.250				0.250
17					0.500							
25						0.250	0.250					
28								0.500				
18									0.500			
23									0.250		0.250	
27										0.250		0.250
29												0.500
22												
30												
35												
37												
32												
36												
43												
45												
34												
38												
41												
46												
31												
42												
44												
48												
33	1.000											
40	1.010											
47		1.000										
49		1.010										
39	1.000											
50		1.010										

	31	42	44	48	33	40	47	49	39	50	y
1											0
2											0
3											0.01
4											0
6											0.01
7											0.02
5											0
8											0.01
10											0.01
13											0.02
9											0
12											0.01
14											0.01
15											0.02
11											0
16											0.01
20											0
19											0.01
21											0.01
24											0.02
26											0.02
17	0.500										0
25		0.250	0.250								0.01
28				0.500							0.02
18	0.250				0.250						0
23		0.250				0.250					0.01
27			0.250				0.250				0.01
29				0.250				0.250			0.02
22					0.250	0.500			0.250		0
30							0.500	0.250		0.250	0.01
35											0.02
37											0.02
32											0.01
36											0.01
43											0.02
45											0.02
34											0.01
38											0.01
41											0.02
46											0.02
31											0
42											0.01
44											0.01
48											0.02
33											0
40											0.01
47											0.01
49											0.02
39											0
50											0.01