An Efficient Solution Technique for a Class of Markov Chains Models *

M. Meo[1], E. de Souza e Silva[2], M. Ajmone Marsan[1],
[1] Dipartimento di Elettronica, Politecnico di Torino – Italy
[2] Federal University of Rio de Janeiro, NCE and CS Dept. – Brazil

1. Introduction

Performability analysis is traditionally based upon a variety of performance/dependability measures that can be derived from models in which rewards are assigned to states and/or transitions of the system being analysed. Most approaches used for the computation of performability measures are based on Markovian analysis, exploiting transient or steady-state solution techniques, depending on the final measure being calculated. In this work we are concerned with the steady-state solution of Markov chain models. In particular, we propose a method that proved to be very efficient for a class of interesting models in the domain of queueing and high-speed telecommunication systems.

2. The algorithm

We consider the computation of the steady-state distribution of an ergodic Markov chain with state space cardinality equal to N (where N is typically very large), and transition probability matrix $\mathbf{P}$, $\mathbf{P} = \{p_{i,j}\}_{i,j=0}^{N-1}$. Once this distribution is obtained, several performability measures can be computed, such as the expected long term cumulative reward averaged over time, the mean time to reach a subset of states in the model, the long term fraction of time above a reward level, etc.

The steady-state distribution of the Markov chain is given by the row vector $\boldsymbol{\pi} = \{\pi_i\}_{i=0}^{N-1}$.

The computation of the steady-state distribution of the Markov chain requires the solution of the matrix equation $\boldsymbol{\pi}\mathbf{P} = \boldsymbol{\pi}$, and the normalization of vector $\boldsymbol{\pi}$ (or, equivalently, the solution of $\boldsymbol{\pi}\mathbf{Q} = \mathbf{0}$, where $\mathbf{Q}$ is the transition rate matrix of a continuous time Markov chain model.)

Most direct solution methods for linear equations are based on the successive elimination of variables from the linear system. For instance, the GTH algorithm [1] eliminates variables using diagonal pivoting like the usual Gaussian elimination, but at each step a clever modification is performed in the multipliers to avoid subtractions, taking advantage of the fact that the row sum of a stochastic matrix is equal to one. The method we use is also a slight modification of the classical Gaussian elimination approach.

We proceed by working with the flow equations. We rewrite $\boldsymbol{\pi}\mathbf{P} = \boldsymbol{\pi}$ as $\boldsymbol{\pi}\mathbf{A} = \mathbf{0}$, with $\mathbf{A} = \{a_{i,j}\}_{i,j=0}^{N-1} = (\mathbf{I} - \mathbf{P})$. We thus obtain the homogeneous system of linear equations:

$$\sum_{i=0}^{N-1} \pi_i \, a_{i,j} = 0, \quad j = 0 : N-1 \tag{1}$$

We solve the first equation (obtained from the first column of $\mathbf{A}$, i.e. from the entries $a_{i,j}$ with $j = 0$):

$$\sum_{i=0}^{N-1} \pi_i \, a_{i,0} = 0$$

for π_1 and we obtain:

$$\pi_1 = -\pi_0 \frac{a_{0,0}}{a_{1,0}} - \sum_{i=2}^{N-1} \pi_i \frac{a_{i,0}}{a_{1,0}}$$

assuming that $a_{1,0} \neq 0$.

It is now possible to eliminate π_1 from all other equations, thus obtaining a reduced system of linear equations:

$$\pi_0 \left(a_{0,j} - \frac{a_{0,0} a_{1,j}}{a_{1,0}} \right) + \sum_{i=2}^{N-1} \pi_i \left(a_{i,j} - \frac{a_{i,0} a_{1,j}}{a_{1,0}} \right) = 0 \qquad j = 1 : N-1$$

This reduced system of linear equations has the form $\boldsymbol{\pi}\mathbf{A}^{(1)} = \mathbf{0}$, where now $\mathbf{A}^{(1)}$ is an $(N-1) \times (N-1)$ matrix.

The same procedure can now be applied to matrix $\mathbf{A}^{(1)}$, for a further reduction of the linear system size.

*This work was supported in part by a research contract between Politecnico di Torino and CSELT, in part by the EC through the Copernicus project 1463 ATMIN, and in part by the Italian Ministry for University and Research. E. de Souza e Silva was supported in part by CNPq grants 524003/94-7 and 680072/95-0.

The algorithm is recursively applied until $N-1$ variables are eliminated. The i-th step of the algorithm consists in solving equation i (obtained from column $i-1$ of $\mathbf{A}$) for π_i and eliminating π_i from all the equations that have not yet been solved.

Since $\mathbf{A}$ is singular, we set π_0 to 1, and compute the other π_i's from the flow equations (see step 5 of the algorithm below). Vector $\boldsymbol{\pi}$ is then normalized at the end of the procedure.

In the implementation of the algorithm, in order to minimize memory occupancy, only one data structure for the storage of an $N \times N$ matrix can be used. Matrix $\mathbf{A}$ is initially stored in such a data structure, and all the reduced matrices that are generated at the subsequent steps of the solution algorithm can be overwritten in the same data structure. Denoting with $\mathbf{A}^{(k)} = \{a_{ij}^{(k)}\}$ the matrix resulting at step k of the procedure, we formally describe the algorithm as follows:

1. Set step k to 0 and initialize the matrix: $\mathbf{A}^{(k)} = \mathbf{A}$
2. Compute matrix $\mathbf{A}^{(k+1)}$ from $\mathbf{A}^{(k)}$ as follows:
$$a_{i,k}^{(k+1)} = -a_{i,k}^{(k)} / a_{k+1,k}^{(k)}$$
$$a_{i,j}^{(k+1)} = a_{i,j}^{(k)} + a_{i,k}^{(k+1)} a_{k+1,j}^{(k)}$$
for all $i = 0 \cup k+1 \leq i \leq N-1$ and $k+1 \leq j \leq N-1$.
3. Increment k and, if $k < N-1$, go to 2
4. Set π_0 to 1, $k = N-1$
5. Compute π_k:
$$\pi_k = \pi_0 a_{0,k-1}^{(k)} + \textstyle\sum_{i=k+1}^{N-1} \pi_i a_{i,k-1}^{(k)}$$
6. Decrement k and, if $k > 0$, go to 5
7. Normalize state vector $\boldsymbol{\pi}$

This algorithm can be easily extended to cope with block matrices. Consider a partition of the steady-state probability vector $\boldsymbol{\pi}$ into B sub-vectors, $\{\boldsymbol{\pi}_0, \boldsymbol{\pi}_1, \cdots, \boldsymbol{\pi}_{B-1}\}$. In order to simplify the notation we assume that all partitions have identical size D. Matrix $\mathbf{A}$ is subdivided accordingly into B^2 sub-matrices $\mathbf{A}_{i,j}$ of dimension $D \times D$.

Equation (1) can now be rewritten as:

$$\sum_{i=0}^{B-1} \boldsymbol{\pi}_i \mathbf{A}_{i,j} = 0, \quad j = 0 : B-1 \tag{2}$$

and, assuming that $\mathbf{A}_{1,0}$ can be inverted, we can solve it for $\boldsymbol{\pi}_1$:

$$\boldsymbol{\pi}_1 = -\boldsymbol{\pi}_0 \mathbf{A}_{0,0} \left(\mathbf{A}_{1,0}\right)^{-1} - \sum_{i=2}^{B-1} \boldsymbol{\pi}_i \mathbf{A}_{i,0} \left(\mathbf{A}_{1,0}\right)^{-1}$$

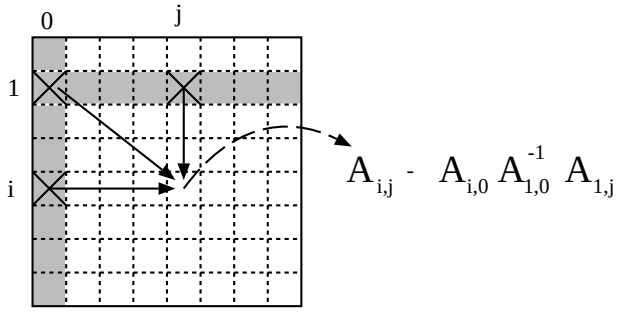

Figure 1: Derivation of $\mathbf{A}^{(1)}$ from $\mathbf{A}$

Eliminating $\boldsymbol{\pi}_1$ from all the other equations, we get the reduced system of linear equations:

$$\boldsymbol{\pi}_0 \left(\mathbf{A}_{0,j} - \mathbf{A}_{0,0} \left(\mathbf{A}_{1,0}\right)^{-1} \mathbf{A}_{1,j}\right) + \sum_{i=2}^{B-1} \boldsymbol{\pi}_i \left(\mathbf{A}_{i,j} - \mathbf{A}_{i,0} \left(\mathbf{A}_{1,0}\right)^{-1} \mathbf{A}_{1,j}\right) = 0 \qquad j = 1 : B-1$$

which can be written in matricial form as $\boldsymbol{\pi} \mathbf{A}^{(1)} = \mathbf{0}$.

Figure 1 shows how matrix $\mathbf{A}^{(1)}$ is derived from $\mathbf{A}$. The shaded column refers to the matrix equation used for the elimination of $\boldsymbol{\pi}_1$. Block $\mathbf{A}_{i,j}$ of $\mathbf{A}$ is transformed into $\mathbf{A}_{i,j}^{(1)}$ according to the formula given in the figure, which involves a block of the shaded row and column. The formula requires the inversion of $\mathbf{A}_{1,0}$, which, therefore, has to be non-singular.

If we assume that all $\mathbf{A}_{k+1,k}^{(k)}$ are invertible, we can describe our algorithm as follows:

1. Set step k to 0 and initialize the matrix: $\mathbf{A}^{(k)} = \mathbf{A}$
2. Compute matrix $\mathbf{A}^{(k+1)}$ from $\mathbf{A}^{(k)}$ as follows:
$$\mathbf{A}_{i,k}^{(k+1)} = -\mathbf{A}_{i,k}^{(k)} \left(\mathbf{A}_{k+1,k}^{(k)}\right)^{-1}$$
$$\mathbf{A}_{i,j}^{(k+1)} = \mathbf{A}_{i,j}^{(k)} + \mathbf{A}_{i,k}^{(k+1)} \mathbf{A}_{k+1,j}^{(k)}$$
for all $i = 0 \cup k+1 \leq i \leq B-1$ and $k+1 \leq j \leq B-1$.
3. Increment k and, if $k < B-1$, go to 2
4. Derive $\boldsymbol{\pi}_0$ from $\mathbf{A}_{0,B-1}^{(B-1)}$
5. Compute $\boldsymbol{\pi}_k$:
$$\boldsymbol{\pi}_k = \boldsymbol{\pi}_0 \mathbf{A}_{0,k-1}^{(k)} + \textstyle\sum_{j=k+1}^{B-1} \boldsymbol{\pi}_j \mathbf{A}_{j,k-1}^{(k)}$$
6. Decrement k and, if $k > 0$, go to 5
7. Normalize state vector $\boldsymbol{\pi}$

Note that at step 4, the algorithm requires the solution of $\boldsymbol{\pi}_0 \mathbf{A}_{0,B-1}^{(B-1)} = \mathbf{0}$.

3. Applications

The GTH algorithm [1] guarantees that all $\mathbf{A}^{(k)}$ are $M-$matrices, and operates with additions and multiplications only. The proposed method does not exhibit these nice properties, and it requires that $\mathbf{A}_{k+1,k}^{(k)} \neq \mathbf{0}$ is invertible. However, despite these disadvantages, it shows a much higher efficiency when dealing with special cases that often occur in the analysis of queueing models and high-speed cell-based telecommunication networks. We refer to the case of matrices with banded structure, that is matrices $\mathbf{A} = \{a_{i,j}\}_{i,j=0}^{N-1}$ in which for some $h < N$ and some $g < N$, $a_{i,j} = 0$ for $j > i+g$ and for $i > j+h$. In order to simplify the explanation we focus on banded matrices where $h = 1$; this is quite an important case from the applications viewpoint. It is easy to see that, like GTH, our algorithm preserves the banded structure of the matrix during the steps of the recursive solution, that is, after each step, the resulting reduced matrix remains banded.

When GTH is applied to this kind of matrices, the number of required element computations at each step of the algorithm grows until the first row fills up to its $(i, i+g)$-th element. From that point on, g elements must be computed at each step. With our approach, like with GTH, the first row of the reduced matrices fills up with nonzero entries. However, it can be easily seen that now the number of element computations at each step depends on the number $r \leq g$ of nonzero elements of the second row of the reduced matrix that are located to the right of the first column. This number depends on the initial structure of the original matrix and is not affected by the reduction procedure. Disregarding the computations during the phase in which the first row fills up with nonzero entries, it is quite easy to see that the numbers of operations required by the two algorithms are $(N-1)(g+1)$ multiplications and $(N-1)g$ sums for GTH and $(N-1)r$ multiplications and $(N-1)(r-1)$ sums for the new algorithm, where N is the number of states in the model. When $h > 1$ similar savings can be obtained.

In the case of block banded matrices, in addition to the savings obtained because of the smaller number of required operations, the proposed method shows another important advantage. This is due to the fact that the computational complexity of the inversion of the elementary block that needs to be inverted at each step is crucial for the determination of the overall algorithm cost. Our method, unlike the GTH algorithm, does not modify the elements of the lower diagonal blocks $\mathbf{A}_{k+1,k}^{(k)}$ of the original matrix $\mathbf{A}$. Operations on blocks often tend to reduce their sparseness, and to destroy any regularity in their structure; if blocks $\mathbf{A}_{k+1,k}$ are unaffected by the algorithm, any special structure initially present in them can be exploited in order to improve the efficiency of the inversion. Moreover, in a number of interesting models, most of the lower diagonal blocks are identical, so that only one matrix inversion is necessary to cope with all the identical blocks. It is important to note that this characteristic contrasts with block GTH. In that algorithm, at each step, a block in the diagonal is modified, and any special structure of the diagonal block is lost.

For matrices with no particular structure, the proposed algorithm has the same computational requirements as GTH, or any Gaussian elimination based algorithm.

4. Examples

As an example of application of the proposed algorithm, we consider the model discussed in [2, 3]. It consists of two finite-capacity queues with 5 servers each. as shown in Figure 2. If a customer arrives at the first queue and no room is available in the waiting line, it moves to the second queue, if any space is left there. If we associate a cost per unit time with each buffer occupied in the first queue, one measure of interest is the long term expected cost averaged over time.

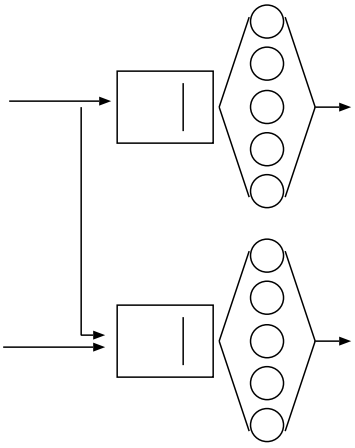

Figure 2: Example application

By assuming that the waiting rooms of both queues have identical capacity $b-1$, the model can be described by an infinitesimal generator with tridiagonal block structure. Each block has size $b \times b$, and the main diagonal of the infinitesimal generator is composed of b blocks.

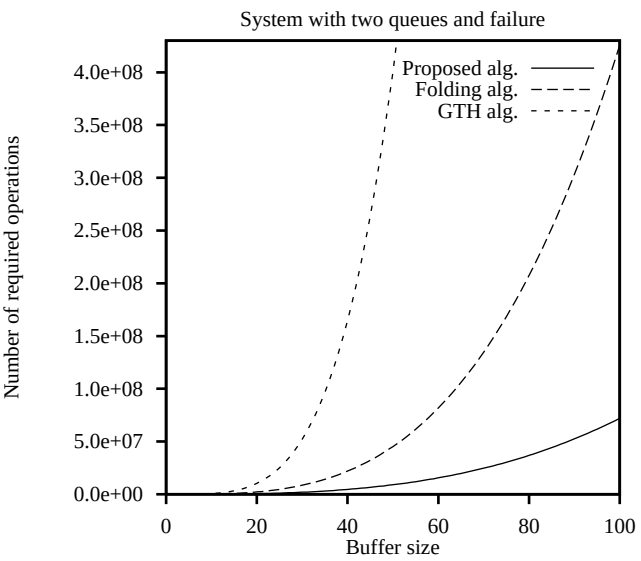

Figure 3: Number of required operations with the proposed method and with the GTH and Folding algorithms versus buffer size, for the 2-queue system with failures

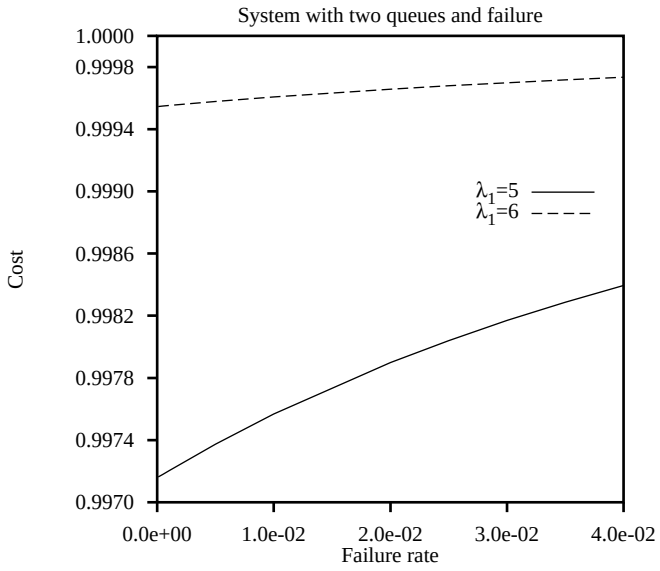

Figure 4: Cost per minute versus the failure rate at the first queue

Consider now a modification of this model that allows each of the two services (set of servers) to fail. Whenever a failure occurs at a queue, the service is stopped at that queue until the failure is repaired. The state variable is (b_1, o_1, b_2, o_2), where b_i is the buffer occupancy at queue i, and o_i indicates whether the servers at queue i are operational or not. This model extension causes the state space cardinality to increase from b^2 up to $4b^2$.

We partition the steady-state probability vector according to the value of b_1, the buffer occupancy at queue 1, therefore we apply our procedure to blocks whose size is $4b \times 4b$. The lower diagonal blocks are diagonal matrices and this structure is exploited by our algorithm.

The computational requirements for the proposed algorithm, for GTH and for the Folding algorithm are shown for this case in Figure 3. The reduction in computational complexity is remarkable, and is mostly due to the fact that each step of the procedure does not affect the blocks to be inverted in the following steps so that their sparseness can be exploited to reduce the number of computations.

In Figure 4 we plot the long term expected cost per minute versus the failure rate at the first queue, assuming that a unitary cost per minute is incurred when the buffer at the first queue is not empty. Two different values are considered for the arrival rate λ_1 at the first queue: $\lambda_1 = 5$ and $\lambda_1 = 6$ customers per minute. The arrival rate λ_2 at the second queue is taken to be one tenth of λ_1. At each queue the number of servers is 5, the waiting room capacity is $b - 1 = 15$, the average service time equals 1 minute, and the mean repair time is about 10 minutes. The time between failures of the second queue is about 2 years.

References

[1] W.K. Grassmann, M.I. Taksar, and D.P. Heyman. Regenerative analysis and steady state distributions for Markov chains. *Operation Research*, 33(5):1107–1116, 1985.

[2] D.P. Heyman. Further comparisons of direct methods for computing stationary distributions of Markov chains. *SIAM J. Alg. Disc. Meth.*, 8(2), April 1987.

[3] L. Kaufman. Matrix methods for queueing problems. *SIAM J. Sci. Stat. Comput.*, 4(3), September 1983.

[4] J. Ye and S.Q. Li. Folding algorithm: A computational method for finite QBD processes with level-dependent transitions. *IEEE Trans. on Communications*, 4:625–639, 1994.