A Comparison of Two Stable Approaches for Computing Steady State Measures for Markov Chains

R. Marie[1], A. Merlo[2], B. Sericola[1]

[1] IRISA – INRIA, Campus de Beaulieu
35042 Rennes Cedex, France
e–mail: {marie,sericola}@irisa.fr
[2] Dipartimento di Informatica e Sistemistica, Università di Pavia
Via Ferrata 1, I–27100 Pavia, Italy
e–mail: merlo@gilda.unipv.it

Abstract. Evaluation studies of computer systems deal often with the analysis of random phenomena that arise when contentions on system resources imply an overall behavior which is not predictable in a deterministic fashion. In these cases, in general, the statistical regularities that are nevertheless present allow the construction of a probabilistic model of the observed system. In this paper we address a performance comparison between two stable approaches for computing some steady state measures for Markov chains. These algorithms reveal particularly suitable when the infinitesimal generator of the Markov chain is ill–conditioned. Our analysis is carried out by means of a few numerical case studies, dealing with different structures and different dimensions of the infinitesimal generators themselves.

Keywords – Markov chains, steady state, queuing system, uniformization

1 Introduction

Evaluation studies of computer systems deal often with the analysis of random phenomena that arise when contentions on system resources imply an overall behavior which is not predictable in a deterministic fashion. Conflicts among transactions in a data base environment and collisions between the transmissions issued by different systems sharing a single communication channel are only a few examples of this situation. In these cases, in general, the statistical regularities that are nevertheless present allow the construction of a probabilistic model of the observed system.
When a system is modeled by an irreducible continuous time Markov chain (CTMC), there is often an interest in computing a steady state measure which can be expressed as a linear function of the steady state probability distribution. For example, we may be interested in the steady-state loss probability of a M/PH/1/K queue, or in the asymptotic expectation of the random reward function in a fault tolerant architecture.
In such a situation, the standard approach is to obtain the steady state probability distribution through the resolution of a linear system of equations and then to compute the steady state measure of interest. There exists algorithms that address the resolution of the linear system of equations from a general viewpoint. On the other hand, there exists algorithms that use the information that we are looking for the steady state behavior of a Markov chain.
Two major aspects characterize such type of algorithms: the size of the state space and the possible ill-conditioning of the matrix that describes the underlying stochastic process. The former determines the complexity in time of the resolution methods and is responsible of the memory requirements to attain the desired results. A number of different algorithms, either direct or iterative, have been proposed in the literature to avoid such problems (see, for example, [5]). The latter aspect is connected to the numerical stability achieved by the algorithm. With respect to the same CTMC, some algorithms behave better

than others. For example, the GTH algorithm [1] has become very popular in the last decade. Indeed, this algorithm works only on non-negative real values and does not use any subtractions at all.

This paper is organized as follows. Two state–of–the–art methods to derive the probability vectors of Markov chains are presented in Section 2. In particular, the basic ideas underlying the Sheskin's algorithm (see Section 2.1) and the GTH approach (see Section 2.2) are described and it is shown that these methods are equivalent. Section 3 presents a new iterative technique, precisely, the Steady State Detection (SSD) algorithm, to compute some steady state measures for Markov chains. A comparative study between the performance attained by the new iterative algorithm with respect to the GTH/Sheskin's approach is outlined in Section 4. Such an analysis is carried out by means of a few numerical case studies, dealing with different types of Markov chains. Finally, a few conclusions are drawn in Section 5.

2 Stable Algorithms for Steady State Computation

The basic characteristics of two methods that have been widely used to compute the steady state distributions of Markov chains are pointed out in the following sections. In particular, after a brief presentation of the underlying algorithms, we will show that these methods are equivalent.

2.1 The Sheskin's Algorithm

The Sheskin's algorithm [4] is a method to compute the steady state distribution by means of iterate partitions of the state space $S = \{1, \ldots, N\}$.
Let $P = [p_{i,j}]$, $i, j \in S$, be the transition probability matrix of an irreducible Markov chain over the state space S. We denote by π the steady state distribution vector of the Markov chain so that π verifies $\pi = \pi P$. As a first step, the state space is partitioned as $S = \{1, \ldots, N-1\} \cup \{N\}$ and the matrix P is accordingly decomposed over this partition as

$$P = \left(\begin{array}{c|c} T & W \\ \hline R & Q \end{array}\right)$$

where T is a matrix of dimension $(N-1) \times (N-1)$, W is a column vector of dimension $(N-1)$, R is a row vector of dimension $(N-1)$ and Q is the scalar $p_{n,n}$.

The matrix P' of dimension $(N-1)$ defined by

$$P' = T + W(1-Q)^{-1}R$$

is an irreducible stochastic matrix and we denote by π' its steady state distribution, that is π' verifies $\pi' = \pi' P'$. If we write $\pi = (x, \pi_N)$, where x is a row vector of dimension $(N-1)$, then

$$\pi_N = xW + \pi_N Q$$

that is

$$\pi_N = xW(1-Q)^{-1}.$$

which involves that x verifies the relationship $x = xP'$. Therefore, the vectors π' and x are proportional. We denote by c the coefficient of proportionality between x and π' so that $x = c\pi'$. As a result, $c = 1 - \pi_N$ and thus

$$\pi_N = \frac{\pi' W(1-Q)^{-1}}{1 + \pi' W(1-Q)^{-1}} \text{ and } x = (1-\pi_N)\pi'.$$

It follows that π can be computed if π' is known. Note that, once the matrix P' has been computed, the vector R is no longer needed.

By applying $(N-2)$ times the same partitioning approach, starting from matrix P', we can compute the whole steady state distribution vector of the original Markov chain. The overall algorithm (for more details see [4]) can be sketched out as follows.

for $n = N$ **to** 2 **do**
    $H = 1 - p_{n,n}$
    $p_{i,n} = p_{i,n}/H$, $1 \leq i < n$
    $p_{i,j} = p_{i,j} + p_{i,n}p_{n,j}$, $1 \leq i,j < n$
endfor

$\sigma = 1$, $r_1 = 1$

for $j = 1$ **to** N **do**
    $r_j = \sum_{k=1}^{j-1} r_k p_{k,j}$
    $\sigma = \sigma + r_j$
endfor

for $j = 1$ **to** N **do**
    $\pi_j = r_j/\sigma$
endfor

Table 1. The Sheskin's algorithm.

Note that, in order to avoid subtractions, the value of H can be computed as $H = \sum_{j=1}^{n-1} p_{n,j}$. Note also that at the end of the loop over n, we must have $p_{1,1} = 1$.

2.2 The GTH Algorithm

The GTH algorithm has been proposed in [1]. It is based on the same partitioning method proposed by Sheskin but it applies to continuous time Markov chains. We will show here that the two algorithms are equivalent.
Consider the infinitesimal generator $A = [a_{i,j}]$, $i,j \in S$, of an irreducible CTMC over the state space S and define the stochastic matrix $P = I + A/\nu$, where I denotes the identity matrix and $\nu \geq \max_{i \in S}\{-a_{i,i}\}$.
The matrix P is the uniformized transition probability matrix associated to the continuous time CTMC [3]. As a result

$$\pi A = 0 \iff \pi = \pi P.$$

As in Sheskin's algorithm, we decompose matrix A over the partition $S = \{1, \ldots, N-1\} \cup \{N\}$ of the state space as

$$A = \left(\begin{array}{c|c} B & C \\ \hline D & E \end{array}\right).$$

Then we define the matrix A' of dimension $(N-1)$ to be

$$A' = B - CE^{-1}D.$$

It is easy to verify that the matrix P' of the previous section is such that $P' = I + A'/\nu$ and that $\pi' A' = 0$. We also have $W(1-Q)^{-1} = -CE^{-1}$.

This shows that the two algorithms are equivalent. To obtain the GTH algorithm from the Sheskin's one, it is sufficient to replace each $p_{i,j}$ with $a_{i,j}$ and to set $H = -a_{n,n}$. Note that in order to avoid subtractions, it is possible to compute first the values of $a_{i,j}$ for $j \neq i$ and then to set $a_{i,i} = -\sum_{j \neq i} a_{i,j}$. Note that at the end of the loop over n, we must have $a_{1,1} = 0$, as similarly we got $p_{1,1} = 1$ before.
Since this equivalence holds, and the two algorithms have been published in the same year, in the following we will refer to them as to a unique method, called SGTH.
The time complexity of this method is given by $O(\frac{2}{3}N^3)$. This order of magnitude has to be compared to the one characterizing the SSD approach, presented in the next section, in order to interpret the results obtained in our comparative study.

3 The Steady State Detection Algorithm

Consider an irreducible CTMC over the state space $S = \{1, \ldots, N\}$ with infinitesimal generator $A = [a_{i,j}]$, $i, j \in S$. Consider again the stochastic matrix $P = I + A/\nu$, where I denotes the identity matrix and ν is chosen such that $\nu \geq \max_{i \in S}\{-a_{i,i}\}$. The matrix P is irreducible and by choosing ν such that $p_{i,i} > 0$ for some i, it becomes ergodic, that is, $\lim_{n \longrightarrow \infty} p^n_{i,j} = \pi_j$ for any value of i. Thus, in what follows, we suppose that the matrix P is ergodic.
The steady state distribution π of the CTMC verifies $\pi A = 0$ or, equivalently, $\pi P = \pi$.

Let $f = \pi\beta$ be a general steady state measure, where β is a column vector with non negative entries. For $n \geq 0$, consider the sequence of column vectors V_n defined by

$$V_n = P^n \beta.$$

As a consequence of the ergodicity of the matrix P, we have

$$f = \lim_{n \longrightarrow \infty} V_n(i), \quad \forall i \in S.$$

For $n \geq 0$, we define the two sequences of real numbers m_n and M_n as

$$m_n = \min_{i \in S} V_n(i) \text{ and } M_n = \max_{i \in S} V_n(i).$$

Theorem 3.1 *The sequences m_n and M_n are respectively non decreasing and non increasing. They both converge to f and, for every $n \geq 0$, we have*

$$|f - \frac{M_n + m_n}{2}| \leq \frac{M_n - m_n}{2}.$$

Proof. See appendix A.
Since both sequences m_n and M_n converge to f, we define, given a fixed error tolerance ε, the integer N_s as

$$N_s = \inf\{n \geq 0 \mid (M_n - m_n)/2 \leq \varepsilon\}.$$

Note that N_s can be interpreted as the discrete time to stationarity with respect to the steady state measure f.

Theorem 3.1 shows that, in order to compute f within an error tolerance equal to ε, we only need to compute the sequence of vectors V_n for $n \leq N_s$.
For a given value of ε, this algorithm can be summarized as follows.

$$n = 0, \quad V_0 = \beta, \quad m_0 = \min_{i \in S} \beta_i, \; M_0 = \max_{i \in S} \beta_i$$

while $(M_n - m_n)/2 \geq \varepsilon$ **do**
$n = n + 1$
$V_n = PV_{n-1}$
$m_n = \min_{i \in S} V_n(i)$
$M_n = \max_{i \in S} V_n(i)$
endwhile

$$g = \frac{M_n + m_n}{2} \qquad (\text{with } |f - g| \leq \varepsilon)$$

Table 2. The SSD algorithm.

Assuming that the matrix P is full, the time complexity of our method is $O(N_s N^2)$, where N is the cardinality of the state space. This complexity, compared to the one which characterizes the previous method, shows that, from a performance viewpoint, our approach may reveal successful if $N_s < \frac{2}{3}N$.

4 Comparative study

In order to compare the approaches described in Sections 2.1 and 2.2 above, we have implemented and executed the SSD and the SGTH algorithms on a SUN SPARCstation system. In particular, we have compared the times taken by these methods to solve different Markov chains varying the dimension of the corresponding infinitesimal generator from 64×64 up to 1024×1024. The error tolerance for the SSD algorithm has been fixed to $\varepsilon = 10^{-5}$.

In our first comparative study, the infinitesimal generator has been chosen to be a full matrix, that is, no zero elements are present. The measured execution times, together with the number of iterations requires by the SSD algorithm to attain the selected error tolerance in the computation of π_N, are reported in Table 3.

	SSD		SGTH
N	N_s	Time [sec]	Time [sec]
1024	5	357.09	1276.40
512	2	83.86	197.98
256	2	21.62	35.58
128	6	5.34	6.79
64	9	1.37	1.53

Table 3. Comparison between SSD and SGTH algorithms in the case of full matrices.

In such a case, we observe that the SSD algorithm outperforms the SGTH one over the whole range of dimensions of the infinitesimal generator. Furthermore, the bigger the infinitesimal generator, the bigger

the gain in the execution time if the SSD algorithm is used. Indeed, we have a 10.45% improvement with 64 states, a 39.23% gain with 256 states, and a 72.02% improvement when matrices of order 1024 are solved.

In the second the infinitesimal generator used to compare the two approaches is derived from a M/PH/1/K process with a two–phases service distribution, that is a block birth–death process. The measure of interest is, in this case, the steady state loss probability.

The number of strictly–positive elements in such an infinitesimal generator is approximatively equal to 3N, where N = 2K + 1 is the order of the infinitesimal generator itself. The measured execution times for the solution of such a system by means of the two approaches are reported in Table 4.

		SSD		SGTH2	SGTH
N	K	N_s	Time [sec]	Time [sec]	Time [sec]
1025	512	6442	183.68	87.28	1028.99
513	256	3776	49.07	22.48	142.39
257	128	2320	13.93	5.83	21.15
129	64	1484	4.14	1.54	5.84
65	32	918	1.24	0.46	1.31

Table 4. Comparison between SSD and SGTH algorithms in the case of a block birth–death process.

As it can be seen, the situation reverts if compared to the first case study if we consider the SGTH2 algorithm, which performs better than the SSD counterpart. As a matter of fact, we have slightly modified the basic SGTH algorithm, as previously used, to add knowledge of the fact that the considered infinitesimal generators contain a great number of zero elements outside the diagonal blocks. Indeed, it is obvious that when we apply the relationship

$$a_{ij} := a_{ij} + a_{in}\frac{a_{nj}}{|a_{nn}|} \quad 1 \leq i, j < n$$

to add the contributions of elements a_{in} and a_{nj} to a_{ij}, such an operation is meaningful if and only if both a_{in} and a_{nj} are not null. As a result, entire rows and columns can be left unchanged if one of the above conditions does not hold, thus avoiding a great number of arithmetic operations to be performed. On the other hand, the advantage that the SSD algorithm can take out of the structure of the matrix is the possibility of using a compact storage scheme. Indeed, this algorithm does not suffer from the odd side fill–in problem. It seems, however, that such a gain is not sufficient to outperform the SGTH2 approach, due to the high number of iterations required by the SSD algorithm to attain an error tolerance equal to ε. However, if we do use the standard SGTH algorithm, and compare its execution times to those of the SSD approach, is the latter which once more performs better (see Table 4).

In order to further compare the SSD and SGTH approaches, we have considered a third case study, where sparse, randomly–built matrices have been used. To obtain meaningful comparisons with respect to the previous case, such matrices have been derived to approximately contain the same number of non zero elements of those generated by the M/PH/1/K processes. Results for this study, where the value of π_N has been computed, are summarized in Table 5.
As in the previous study we have used the SGTH2 algorithm to take advantage of the sparse structure of the matrix. Note that two different behaviors can be identified: the SGTH2 algorithm clearly performs

	SSD		SGTH2
N	N_s	Time [sec]	Time [sec]
1024	156	91.24	170.55
512	230	24.21	35.31
256	209	6.63	7.24
128	119	1.74	1.73
64	181	0.57	0.48

Table 5. Comparison between SSD and SGTH algorithms in the case of sparse matrices.

better for infinitesimal generators of dimensions 64 (18.75%) while the difference in times is negligible for a 128×128 matrix. On the other hand, the SSD algorithm reveals faster on all the remaining cases (the gain reaches a peak of more than 46.50% with 1024 states).
In this last example, the infinitesimal generator contains a great number of zero elements but they are no longer displaced according to a highly–regular structure. In such a case, we are not guaranteed that we can save a great bunch of arithmetical operations with the SGTH2 algorithm.

5 Conclusions

In this paper we have addressed a performance comparison between two stable approaches for computing some steady state measures for Markov chains. In particular, the different performance obtained by the SSD algorithm and by the SGTH method have been analyzed by means of a few numerical case studies. They cover a complete range of problems, dealing with full and block infinitesimal generators, as well as sparse, randomly–built stochastic matrices. The case studies have been carried out by comparing the times required by these approaches to solve Markov chains with a number of states ranging from 64 to 1024.
As a result, we can point out that, if the basic implementation of the SGTH algorithm is used, the SSD approach always reveals faster. In the case of infinitesimal generators characterized by a highly–regular structure, the *a priori* knowledge of such a structure can be fully exploited within the SGTH algorithm to attain better performance.

References

1. W. K. Grassmann, M. I. Taskar, and D. P. Heyman. Regenerative Analysis and Steady State Distributions for Markov Chains. *Operations Research*, 33(5):1107–1116, 1985.
2. E. Parzen. *Stochastic Processes*. Holden–Day, Inc, San Francisco, 1962.
3. S. M. Ross. *Stochastic Processes*. John Wiley & Sons, New York, 1983.
4. T. J. Sheskin. A Markov Chain Partitioning Algorithm for Computing Steady State Probabilities. *Operations Research*, 33(1):228–235, 1985.
5. W. J. Stewart. *Introduction to the Numerical Solution of Markov Chains*. Princeton University Press, Princeton, 1994.

A Proof of Theorem 3.1

The following demonstration is based on ideas taken from [2].

For every $i \in S$, we have $V_{n+1}(i) = \sum_{j \in S} p_{i,j} V_n(j)$. It follows that $m_n \leq V_{n+1}(i) \leq M_n$ and we get

$$m_n \leq m_{n+1} \text{ and } M_{n+1} \leq M_n. \tag{1}$$

It follows that the sequences m_n and M_n are respectively non decreasing and non increasing. We denote by m and M their respective limits when $n \longrightarrow \infty$.

Writing now $f = \pi V_n = \sum_{i \in S} \pi_i V_n(i)$, we get $m_n \leq f \leq M_n$, and, as a result,

$$|f - \frac{M_n + m_n}{2}| \leq \frac{M_n - m_n}{2}.$$

Let us define $d_n = M_n - m_n$. From equation (1), it follows that

$$0 \leq d_{n+1} \leq d_n.$$

To prove that $m = f = M$, it is sufficient to prove that

$$\lim_{n \longrightarrow \infty} d_n = 0. \tag{2}$$

Consider the subsequence d_{nL} for a fixed integer $L > 0$. In order to prove equation (2), we have to show that

$$\lim_{n \longrightarrow \infty} d_{nL} = 0. \tag{3}$$

Since P is the transition probability matrix of an ergodic Markov chain, there exists an integer L such that

$$p^L_{i,j} > 0, \quad i, j \in S.$$

Let us define $c = \min_{i,j \in S} p^L_{i,j}$. Since $p^L_{i,j} > 0$, we have $0 < c < 1/2$.

Now, since $V_{n+1} = PV_n$, we have

$$V_{(n+1)L} = P^L V_{nL}. \tag{4}$$

Let $r \in S$ and $q \in S$ be such that

$$M_{(n+1)L} = V_{(n+1)L}(r) \text{ and } m_{nL} = V_{nL}(q).$$

Then, from equation (4), we have

$$\begin{aligned}
M_{(n+1)L} &= \sum_{j \in S} p^L_{r,j} V_{nL}(j) \\
&= p^L_{r,q} V_{nL}(q) + \sum_{j \neq r} p^L_{r,j} V_{nL}(j) \\
&= p^L_{r,q} m_{nL} + \sum_{j \neq r} p^L_{r,j} V_{nL}(j) \\
&\leq p^L_{r,q} m_{nL} + M_{nL} \sum_{j \neq r} p^L_{r,j} \\
&= p^L_{r,q} m_{nL} + M_{nL}(1 - p^L_{r,q}) \\
&= M_{nL} - (M_{nL} - m_{nL}) p^L_{r,q} \\
&\leq M_{nL} - (M_{nL} - m_{nL}) c
\end{aligned}$$

We have shown that

$$M_{(n+1)L} \leq M_{nL} - (M_{nL} - m_{nL})c \tag{5}$$

Let now $r \in S$ and $q \in S$ be such that

$$m_{(n+1)L} = V_{(n+1)L}(r) \text{ and } M_{nL} = V_{nL}(q).$$

Then, from equation (4), we have

$$\begin{aligned}
m_{(n+1)L} &= \sum_{j \in S} p^L_{r,j} V_{nL}(j) \\
&= p^L_{r,q} V_{nL}(q) + \sum_{j \neq r} p^L_{r,j} V_{nL}(j) \\
&= p^L_{r,q} M_{nL} + \sum_{j \neq r} p^L_{r,j} V_{nL}(j) \\
&\geq p^L_{r,q} M_{nL} + m_{nL} \sum_{j \neq r} p^L_{r,j} \\
&= p^L_{r,q} M_{nL} + m_{nL}(1 - p^L_{r,q}) \\
&= m_{nL} + (M_{nL} - m_{nL}) p^L_{r,q} \\
&\geq m_{nL} + (M_{nL} - m_{nL})c
\end{aligned}$$

We have shown that

$$m_{(n+1)L} \leq m_{nL} + (M_{nL} - m_{nL})c \tag{6}$$

Subtracting equations (5) and (6), we get

$$M_{(n+1)L} - m_{(n+1)L} \leq (M_{nL} - m_{nL})(1 - 2c),$$

and then

$$d_{(n+1)L} \leq d_{nL}(1 - 2c).$$

As a result

$$d_{nL} \leq (1 - 2c)^n d_N,$$

which tends to 0 as n tends to ∞. □