On the exact simulation of functionals of stationary Markov chains [1]

Jean-Marc Vincent [a] Corine Marchand [a]

[a] *Projects APACHE-INRIA and DECORE-IMAG, Laboratoire ID-IMAG, 51, av. Jean Kuntzmann, 38330 Montbonnot, FRANCE*

Abstract

In performance evaluation domain, simulation is an alternative when numerical analysis fail. To avoid the burn-in time problem, this paper presents an adaptation of the perfect simulation algorithm [10] to finite ergodic Markov chain with arbitrary structure. Simulation algorithms are deduced and provide samplings of functionals of the steady-state without computing the state coupling, it speeds up the algorithm by a significant factor. Based on a sparse representation of the Markov chain, the aliasing technique improves highly the complexity of the simulation. Moreover, with small adaptations, it builds a transition function algorithm that ensures coupling.

Key words: Markov chain simulation, perfect simulation, steady-state analysis.

1 Introduction

Markov chain simulation is of fundamental importance in performance evaluation of parallel or distributed computer systems, communication networks, production systems, and so on. In most cases, the global system is modeled by a stochastic automata with probabilistic transition rules. Many formalisms such as queuing networks, stochastic Petri nets, stochastic automata networks and process algebras lead to such a presentation.

To obtain performance indexes, cost functions are defined and could be viewed as applications from the state-space of the system onto a simpler cost space. For

Email addresses: Jean-Marc.Vincent@imag.fr (Jean-Marc Vincent), Corine.Marchand@imag.fr (Corine Marchand).

[1] Research supported in part by DECORE-IMAG project and ACI Sure-Path

example, we may be interested in the throughput of a protocol, a utilization factor in an operating system or a number of customers in a queuing network.

Formally, the evolution of the system is described by a stochastic process $\{X_n\}_{n\in\mathbb{Z}}$ which takes values in a discrete state-space $\mathcal{X}$ which structure could be complex (no geometric considerations could apply). From modeling considerations, the state space is supposed finite of size N.

In most cases in performance evaluation, it is assumed that the stochastic process is a homogeneous and irreducible Markov chain. In that case the dynamics of the system is captured by the transition matrix P. To observe the long run behavior of the system and compute the performance indices the method is to compute the steady state probability vector π that satisfies the linear system

$$\pi P = \pi. \tag{1}$$

After the computation of the steady-state distribution, we calculate the average cost of the system as the expectation of the cost function C on the state space with the steady state distribution. Let

$$\hat{C} = \mathbb{E}_\pi C = \sum_{x\in\mathcal{X}} C(x)\pi(x). \tag{2}$$

The complexity of the average cost function computation mainly depends on the complexity of solving the linear system (1). Computations of costs (2) have a linear complexity in N, which is negligible compared to the complexity of the estimation of π.

When the system can not be solved analytically, by for example reversibility arguments, product forms methods, etc, numerical methods [13] could be used when the state space of the system is not too large. For example, in the context of stochastic automata networks [5], systems with state space size up to $N = 10^7$ states have been solved. However, these numerical methods take into account the structure of the Markov chain and use a compact storage of matrix and vectors.

Another approach, software simulation, consists of the emulation of the behavior of the system by algorithms and measurements on typical trajectories. According to ergodic assumptions, statistical estimations are developed. Then the average cost function is estimated by an integral on the trajectory

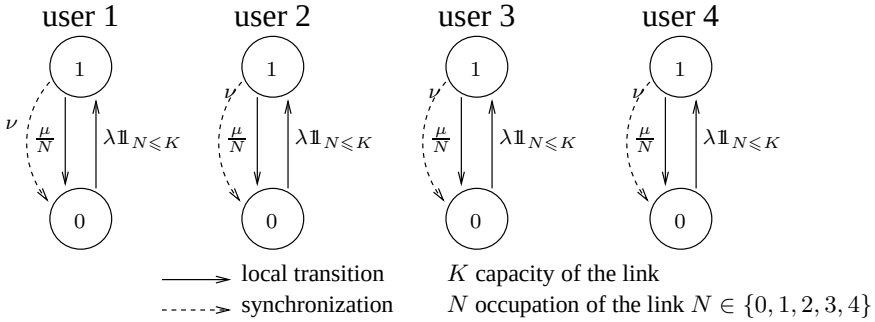
$$\overline{C} = \lim_{n\to+\infty} \frac{1}{n} \sum_{i=0}^{n-1} C(x_i); \tag{3}$$

where $\{x_n\}_{n\in\mathbb{Z}}$ is the sequence of observed states.

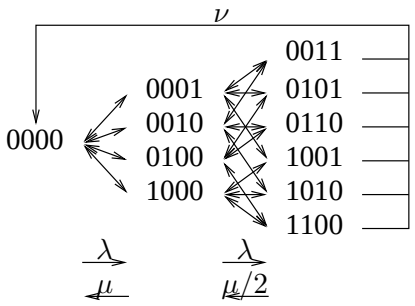
To illustrate this simulation approach, we consider a simple example of a communication system derived from an Erlang model.

Example 1 : Resource sharing model with reset (figure 1)

Several users, say n, may access a communication link. A user is characterized by a think time, modeled by an exponential distribution with rate λ and a communication duration, modeled by an exponential distribution with rate μ. Communications are multiplexed on this link with a maximum of K simultaneous communications. To avoid starvation, a time-out resets the communication link. When the link is saturated, K users are on the link, a timer is armed and after a time-out the link is automatically freed, users are rejected. The time-out is supposed to be exponentially distributed with rate ν. Such a system is easily described by a stochastic automata network [6]. The state space is a n-tuple describing the state of each user (0 thinking and 1 communicating).



Stochastic automata network model



Derived Markov chain

Fig. 1. Model of the resource sharing system with reset, $n = 4$ users, $k = 2$ capacity

One should notice that this kind of Markov chain is neither reversible nor monotone. Its lumpability property is not necessary in the paper, and arbitrary values of coefficients could have been chosen without altering the simulation method.

To simulate and to control the estimation errors, it is generally supposed that the system is stationary and confidence interval methods could be used.

To get statistical informations on the process, the analysis is driven in two steps. In the first step we simulate the system and wait until we consider that it reaches its stationary regime (figure 2). The duration τ of this step is called the burn-in time

of the simulation. It is of importance because the observation depends on the initial state of the simulation; after time τ, the behaviour of the process should not depend on the initial state. After this moment, because of the stationarity assumption, the second step builds statistical estimations on the process based on equation (3).

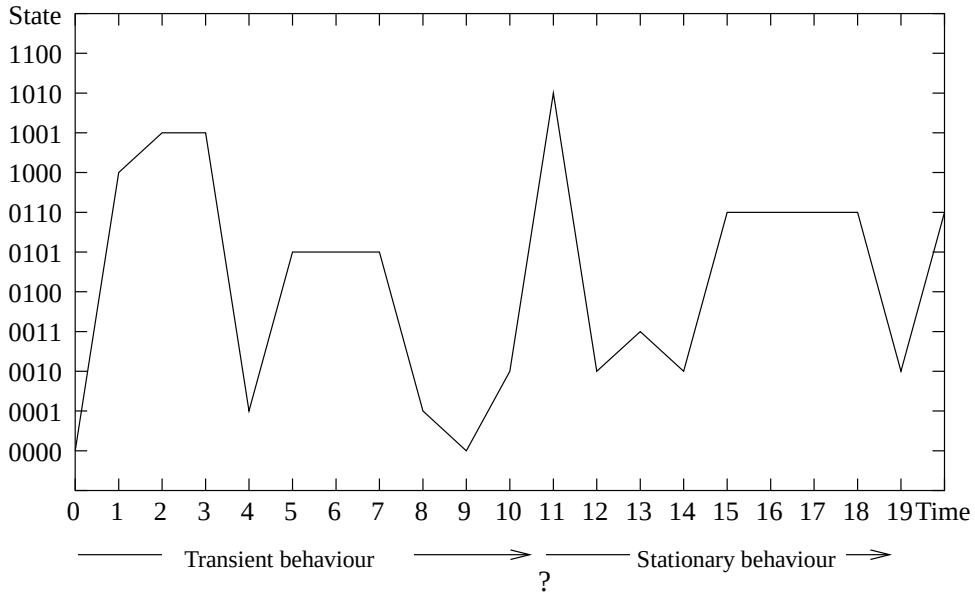


Fig. 2. At which time the chain could be considered stationary ?

Then the main difficulty for this simulation method is to fix the burn-in time τ [1]; for an arbitrary Markov chain, theoretical bounds are loose to give efficient control algorithms. Alternative methods have been developed by Propp and Wilson [10] and avoid this problem by sampling a random variable directly from the unknown stationary distribution π.

Moreover perfect sampling generates independent realizations according to the stationnary distribution. Because of independance, confidence interval techniques could be applied without any bias. There are no correlations inside the sample as in the equation 3 where sampling is made on a stationary sequence where variates are asymptotically normal.

The aim of this article is to adapt the Propp and Wilson technique to a general framework of finite homogeneous and irreducible Markov chains and directly sample cost functions of systems as $C(X)$ where X is sampled according to π. This method has been implemented for sparse matrices in the context of stochastic automata networks. In this paper, we suppose given a transition matrix P which is irreducible and aperiodic, so that all states are positively recurrent.

In the second section we detail the perfect sampling method with a special focus on the representation of the chain by an iterative system of functions. The third section deals with implementation considerations and analyses the complexity of the

simulation program. Finally, the fourth section analyses the efficiency of functional coupling on typical examples issued from performance evaluation.

2 Perfect sampling method

2.1 Iterated systems of functions

Formally, when all the knowledge of the dynamics is included in the state description, the system can be described by a transition function Φ, typically

$$X_{n+1} = \Phi(X_n, U_{n+1}); \tag{4}$$

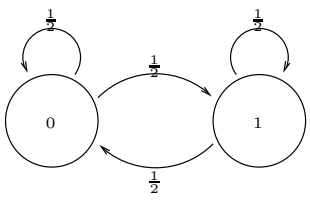
where X_n is n^{th} observed state of the system, and $\{U_n\}_{n\in\mathbb{Z}}$ the sequence of inputs of the system, typically a sequence of calls to a `Random` function. This type of stochastic recursive sequence has been widely studied in a general framework [2] or [4] and some results related with perfect simulation may be found in [11,12].

It is clear that, if the $\{U_n\}$ are independent and identically distributed, the process $\{X_n\}_{n\in\mathbb{Z}}$ defined by an initial value X_0 and the recursive equations (4) is a Markov chain. Conversely, given a transition matrix P, it is possible to find an transition function Φ such that a Markov chain defined by (4) has transition matrix P.

It is important to note that this operator is not unique and a Markov chain could have several representations. For the algorithm design, we have to choose the "best" representation of the Markov chain in terms of complexity reduction. Consider the following example of a two states Markov chain, in which we suppose that the $\{U_n\}$ are uniformly distributed on $[0, 1]$.

Example 2 : Three different representations of a same two state Markov chain

This example, as example 3 further, is derived from the course of Haggstrom [7] chapter 10.



Its transition matrix is

$$P = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} \end{bmatrix},$$

the chain has a unique stationary distribution

$$\pi = (\frac{1}{2}, \frac{1}{2}).$$

The three following operators Φ_a,Φ_b and Φ_c have the same transition matrix

P. These operators have the same value on state 0,

$$\Phi_a(0,u) = \Phi_b(0,u) = \Phi_c(0,u) = \begin{cases} 0 \text{ if } 0 \leqslant u < \frac{1}{2}; \\ 1 \text{ if } \frac{1}{2} \leqslant u < 1. \end{cases}$$

and on state 1 they are defined by

$$\Phi_a(1,u) = \begin{cases} 0 \text{ if } 0 \leqslant u < \frac{1}{2}; \\ 1 \text{ if } \frac{1}{2} \leqslant u < 1. \end{cases}$$

$$\Phi_b(1,u) = \begin{cases} 0 \text{ if } 0 \leqslant u < \frac{1}{4} \text{ or } \frac{3}{4} \leqslant u < 1; \\ 1 \text{ if } \frac{1}{4} \leqslant u < \frac{3}{4}. \end{cases}$$

$$\Phi_c(1,u) = \begin{cases} 0 \text{ if } \frac{1}{2} \leqslant u < 1; \\ 1 \text{ if } 0 \leqslant u < \frac{1}{2}. \end{cases}$$

This example shows that the choice of representation is fundamental for memory and efficiency considerations as will be shown in section 3.

From the code of the stochastic transition function Φ, it is easy to build an algorithm that emulate the behavior of the corresponding Markov chain.

Algorithm 1 Forward simulation of a Markov chain

$x \leftarrow x_0$; {choice of the initial value of the process}
repeat
  $u \leftarrow$ Random; {generation of u_{n+1}}
  $x \leftarrow \Phi(x,u)$; {computation of the next state x_{n+1}}
until stopping criteria
return x

At the end of such an algorithm we obtain a value in $\mathcal{X}$ that should follow the stationary distribution π, the approximation error depends on the heuristic used to terminate the simulation.

Provided that the random generator is statistically correct, the sequence of values of variable x is a realization of a Markov chain with transition matrix P. One should note that the behavior of this algorithm does not depend on the representation of the Markov chain. Moreover the complexity of this algorithm, say the number of evaluations of function Φ, is fixed by the duration of the simulation. For example 2, the complexity of the simulation does not depend on the representation.

2.2 Perfect simulation

To avoid the stopping criteria problem Propp and Wilson [10] proposed an algorithm that directly gives a sample according to the stationary distribution. We first consider the scheme defined by the recurent equation (4) with X_0 arbitrarily distributed. An intuitive idea (not so good as shown in example 3), to stop simulation is to consider all possible initial values, observe their trajectories and stop the simulation when they are all in a same state. We say that all trajectories have coupled. The coupling time is the first time when the trajectories are all in the same state, after the coupling time, the trajectories do not depend on the initial state. The recursive expression (4) is applied to each initial state and we denote by $y(x)$ the current value of the trajectory issued from state x.

Algorithm 2 Forward-coupling simulation

for all $x \in \mathcal{X}$ **do**
  $y(x) \leftarrow x$ {choice of the initial value of the vector y, $n = 0$}
end for
repeat
  u $\leftarrow$ Random; {generation of u_{n+1}}
  for all $x \in \mathcal{X}$ **do**
    $y(x) \leftarrow \Phi(y(x), u)$; {computation of the next state of the trajectory issued from x at time $n = 0$}
  end for
until All $y(x)$ are equal
return $y(x)$

An example of a forward-coupling simulation is illustrated by figure 3.

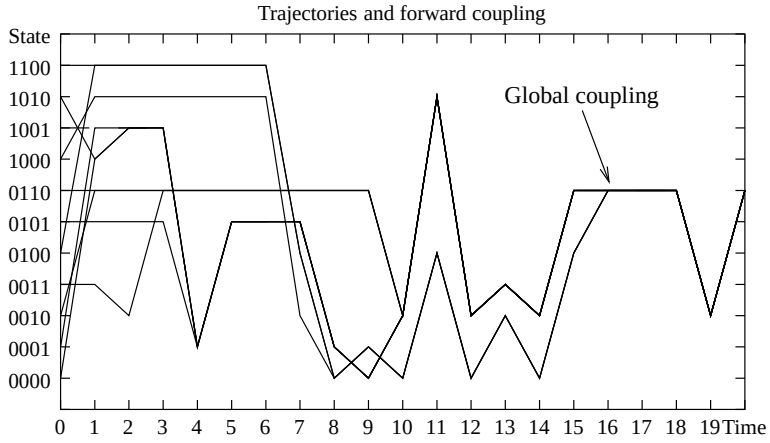
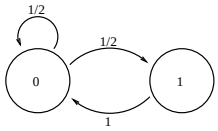


Fig. 3. All trajectories have coupled before time $\tau^* = 16$

A first question is to determine if the algorithm terminated or not. If we denote the coupling time by τ^* , we have to show that $\tau^* < +\infty$ almost surely. Consider the example of the two state chain (example 2). With the Φ_a representation, coupling

occurs at the first step, $\tau^* = 1$. If transitions are coded by Φ_b, a short computation shows that τ^* is geometrically distributed with parameter $\frac{1}{2}$ and is almost surely finite with mean 2. The last representation Φ_c leads to an infinite coupling time so the algorithm never stops !

When the forward coupling algorithm stops all trajectories have coupled, unfortunately the generated state does not follows the stationary distribution. This is clearly illustrated in the example 3.
Example 3 : Coupling in a same state



On the example to the left, it is clear that coupling does not depend of the representation and that coupling time is almost surely finite, geometrically distributed with parameter $\frac{1}{2}$. When 2 trajectories couple, at the preceding step the corresponding states were 0 and 1. But, because the transition probability from 1 to 1 is zero, the trajectories can only couple in 0. Then the generated state is always 0, and is not distributed according to the stationary distribution $\pi = [\frac{2}{3}, \frac{1}{3}]$.

To make this algorithm "exact", Propp and Wilson [10] propose to shift the process in the past. This is equivalent to Loynes [9] monotone scheme used to prove the law convergence of the workload of a queuing system.

Because the sequence $\{U_n\}_{n\in\mathbb{Z}}$ is stationary,

$$\mathbb{P}\left(X_n = j | X_0 = i\right) = \mathbb{P}\left(X_0 = j | X_{-n} = i\right) \tag{5}$$

holds for all n and all couples of states $(i, j) \in \mathcal{X}^2$. Then for any state x and iteration number n

$$\Phi(\Phi(\cdots(\Phi(x, U_1), \cdots), U_{n-1}), U_n) \stackrel{d}{=} \Phi(\Phi(\cdots(\Phi(x, U_{-n+1}), \cdots), U_{-1}), U_0) \tag{6}$$

Provided that the representation of the Markov chain ensures coupling, we modify the algorithm (2) by reversing time leading to algorithm 3:

We illustrate the behavior of this program on our resource sharing system in figure (4).

To understand this algorithm and find conditions of convergence, we consider the sequence of subsets of the state space $\mathcal{X}$, $\{\mathcal{Z}_n\}_{n\in\mathbb{N}}$ defined by

$$\mathcal{Z}_n = \Phi(\Phi(\cdots(\Phi(\mathcal{X}, U_{-n+1}), \cdots), U_{-1}), U_0). \tag{7}$$

Because $\Phi(\mathcal{X}, U_{-n+1}) \subset \mathcal{X}$, we deduce that the sequence $\{\mathcal{Z}_n\}_{n\in\mathbb{N}}$ is non-increasing. Using the finiteness of $\mathcal{X}$ and monotonicity, we deduce that $\{\mathcal{Z}_n\}_{n\in\mathbb{N}}$ converges almost surely to a set $\mathcal{Z}_\infty$. Now we get a new condition for the stopping criteria

Algorithm 3 Backward-coupling simulation

for all $x \in \mathcal{X}$ **do**
  $y(x) \leftarrow x$ {choice of the initial value of the vector y, $n = 0$}
end for
repeat
  u $\leftarrow$ Random; {generation of u_{-n}}
  for all $x \in \mathcal{X}$ **do**
    $y(x) \leftarrow y(\Phi(x, u))$; {computation of the state at time 0 of the trajectory issued from x at time $-n$}
  end for
until All $y(x)$ are equal
return $y(x)$

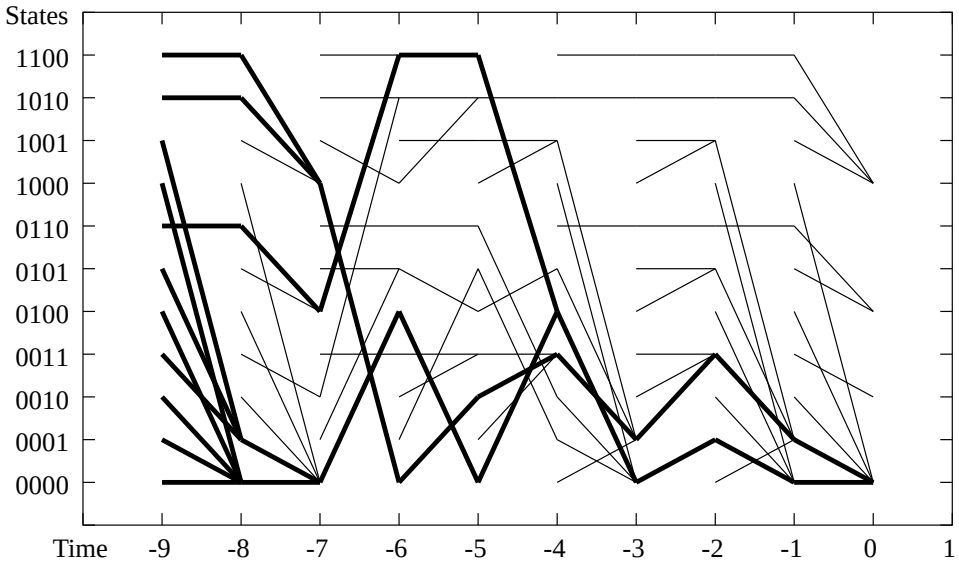


Fig. 4. All trajectories collapsed in state 0000 after 9 steps

Proposition 1 *The algorithm stops, saying $\tau^* < +\infty$ almost surely, if and only if the cardinality of $\mathcal{Z}_\infty$ is 1 almost surely.*

Proof of proposition 1

Suppose that $\tau^* < +\infty$, because algorithms (3) and (2) have the same structure, execution times have the same distribution. And backward coupling time stops almost surely. Because the stopping condition in the algorithm is that all values of $y(.)$ are equal, $Card(\mathcal{Z}_\infty) = 1$.

Conversely, because $\mathcal{X}$ is finite, there exist some time τ for which $Card(\mathcal{Z}_\tau) = 1$. Because τ and τ^* have the same distribution, the conclusion follows.

The main interest of proposition 1 is that it suggests a sufficient condition to check that a representation of a Markov chain leads to a finite coupling time.

Suppose that there exist some set of states $\mathcal{Z}$ of cardinality $k > 1$ such that

$$\mathbb{P}(\Phi(\mathcal{Z}, U) = \mathcal{Z}) = 1. \tag{8}$$

Then, for almost all value u of U, the operator $\Phi(., u)$ is a permutation of $\mathcal{Z}$. The backward and forward process will never couple.

Corollary 1 *Suppose that for each couple of states (x, y) there exists some $n < +\infty$ such that*

$$\mathbb{P}\left[\Phi(\Phi(\cdots(\Phi(x, U_1), \cdots), U_{n-1}), U_n) = \Phi(\Phi(\cdots(\Phi(y, U_1), \cdots), U_{n-1}), U_n)\right] > 0; \tag{9}$$

then $\tau < +\infty$ almost surely.

The proof is simply deduced from proposition 1, if $Card(\mathcal{Z}_\infty) > 1$, Φ is a bijection after some steps, so are the iterations of Φ which is in contradiction with equation (9). Because of the ergodicity of the Markov chain, one could specify the first state x and the condition that every state y couple with x is also sufficient to ensure $\tau < +\infty$. Now, when coupling occurs almost surely, we are able to establish the convergece result.

Proposition 2 *Suppose $\tau < +\infty$ almost surely, (i.e. the algorithm terminates), then the generated value $y(x)$ by algorithm 3 is distributed according to the stationary distribution.*

Proof of proposition 2

This result comes from equation (6). For all $n \geqslant \tau$ and all x we have

$$\Phi(\Phi(\cdots(\Phi(x, U_{-n+1}), \cdots), U_{-1}), U_0) = \Phi(\Phi(\cdots(\Phi(x, U_{-\tau+1}), \cdots), U_{-1}), U_0)$$

Denote by X the value generated by the algorithm and a an arbitrary state.

$$\begin{aligned}
\mathbb{P}(X = a) &= \mathbb{P}\left(\Phi(\Phi(\cdots(\Phi(x, U_{-\tau+1}), \cdots), U_{-1}), U_0) = a\right); \\
&= \mathbb{P}\left[\bigcup_{n=0}^{+\infty} \left(\Phi(\Phi(\cdots(\Phi(x, U_{-\tau+1}), \cdots), U_{-1}), U_0) = a, \tau \leqslant n\right)\right]; \\
&= \mathbb{P}\left[\bigcup_{n=0}^{+\infty} \left(\Phi(\Phi(\cdots(\Phi(x, U_{-n+1}), \cdots), U_{-1}), U_0) = a, \tau \leqslant n\right)\right]; \\
&= \lim_{n \to +\infty} \mathbb{P}\left(\Phi(\Phi(\cdots(\Phi(x, U_{-n+1}), \cdots), U_{-1}), U_0) = a, \tau \leqslant n\right); \\
&= \lim_{n \to +\infty} \mathbb{P}\left(\Phi(\cdots(\Phi(x, U_1), \cdots), U_{n-1}), U_n) = a\right); \\
&= \pi(a).
\end{aligned}$$

This ends the proof.

When a cost function C has to be evaluated on the stationary distribution, the backward simulation algorithm (3) could be adapted to produce directly a sample of $C(x)$. Because all the trajectories converge to the same state, we could stop the algorithm when all trajectories have the same cost. This is just a modification of the initialization of algorithm (3):

Algorithm 4 Functional backward-coupling simulation

for all $x \in \mathcal{X}$ **do**
  $y(x) \leftarrow C(x)$ { choice of the **initial value** of the vector y, at $n = 0$, a trajectory finishing in state x has **cost** $C(x)$}
end for
repeat
  u $\leftarrow$ Random; {generation of u_{-n}}
  for all $x \in \mathcal{X}$ **do**
    $y(x) \leftarrow y(\Phi(x, u))$; {computation of the **cost** of the state at time 0 for the trajectory issued from x at time $-n$}
  end for
until All $y(x)$ are equal
return $y(x)$

It is clear that under the same assumptions, the algorithm stops and the distribution of $C(X)$ is the projection of the stationary distribution π by C.

Consider our example and look at the estimation of resource saturation, saying that 2 users share the resource. The cost function is simply defined by

$state$	$cost$	$state$	$cost$
0000	0	0110	1
0001	0	1000	0
0010	0	1001	1
0011	1	1010	1
0100	0	1100	1
0101	1		

The backward simulation is executed until all trajectories finish on a same cost state. In figure (5) only 3 steps are necessary, which is "much smaller" than the stopping time $\tau^* = 9$. This amount of reduction of complexity could be significant, see for example section 4, reduction by an order of magnitude. But, in some cases, it could be as large as the global coupling time.

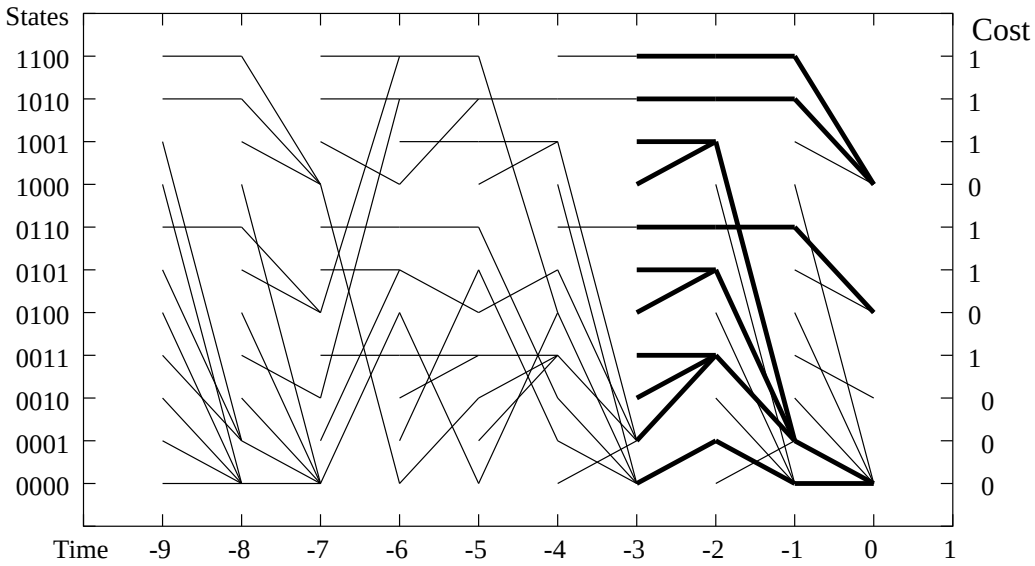


Fig. 5. All trajectories collapsed in state cost 0 after 3 steps

3 Implementation considerations

3.1 Sparse matrix format

We suppose that the transition matrix of the Markov chain is given in some sparse format. The typical representation used in our application is derived from the "Harwell-Boeing Form" as described by Stewart [13].

In our example of shared resources, with the coding of states:

$state$	$code$	$state$	$code$
0000	0	0110	6
0001	1	1000	7
0010	2	1001	8
0011	3	1010	9
0100	4	1100	10
0101	5		

the generator matrix Q is

	0	1	2	3	4	5	6	7	8	9	10
0	-4λ	λ	λ	0	λ	0	0	λ	0	0	0
1	μ	$-(\mu+3\lambda)$	0	λ	0	λ	0	0	λ	0	0
2	μ	0	$-(\mu+3\lambda)$	λ	0	0	λ	0	0	λ	0
3	ν	$\mu/2$	$\mu/2$	$-(\mu+\nu)$	0	0	0	0	0	0	0
4	μ	0	0	0	$-(\mu+3\lambda)$	λ	λ	0	0	0	λ
5	ν	$\mu/2$	0	0	$\mu/2$	$-(\mu+\nu)$	0	0	0	0	0
6	ν	0	$\mu/2$	0	$\mu/2$	0	$-(\mu+\nu)$	0	0	0	0
7	μ	0	0	0	0	0	0	$-(\mu+3\lambda)$	λ	λ	λ
8	ν	$\mu/2$	0	0	0	0	0	$\mu/2$	$-(\mu+\nu)$	0	0
9	ν	0	$\mu/2$	0	0	0	0	$\mu/2$	0	$-(\mu+\nu)$	0
10	ν	0	0	0	$\mu/2$	0	0	$\mu/2$	0	0	$-(\mu+\nu)$

In this case we consider the discrete time Markov chain obtained by uniformization of the matrix Q. That is $P = Id + \frac{1}{\Lambda_{max}}Q$, with Λ_{max} the maximum of the absolute values of diagonal elements of Q. As an example, fixing values of $\lambda = 0.7$, $\mu = 1$ and $\nu = 10^{-2}$, we get $\Lambda_{max} = 3.1$ and the matrix P :

	0	1	2	3	4	5	6	7	8	9	10
0	0.096	0.226	0.226	0	0.226	0	0	0.226	0	0	0
1	0.323	0	0	0.226	0	0.226	0	0	0.226	0	0
2	0.323	0	0	0.226	0	0	0.226	0	0	0.226	0
3	0.003	0.161	0.161	0.674	0	0	0	0	0	0	0
4	0.323	0	0	0	0	0.226	0.226	0	0	0	0.226
5	0.003	0.161	0	0	0.161	0.674	0	0	0	0	0
6	0.003	0	0.161	0	0.161	0	0.674	0	0	0	0
7	0.323	0	0	0	0	0	0	0	0.226	0.226	0.226
8	0.003	0.161	0	0	0	0	0	0.161	0.674	0	0
9	0.003	0	0.161	0	0	0	0	0.161	0	0.674	0
10	0.003	0	0	0	0.161	0	0	0.161	0	0	0.674

To this matrix P, we associate the following data structure with 45 non-null elements. The row index array (index of the first element of the row in the value array, the last index is the total number of non-null elements):

	0	1	2	3	4	5	6	7	8	9	10	11
Index of row	0	5	9	13	17	21	25	29	33	37	41	45

The column index array and the corresponding value V :

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	...
Column	0	1	2	4	7	0	3	6	9	0	1	2	3	0	...
Value	0.096	0.226	0.226	0.226	0.226	0.323	0.226	0.226	0.226	0.003	0.161	0.161	0.674	0.323	...

In this example, elements of row 2 begins at index 9, saying the element at row 2 and column 0 has value 0.03

3.2 Walker's algorithm modified

The aliasing technique, designed by Walker [14], provides an efficient method to build a $\Phi(x, .)$ function, that simulates the next state following x according to the

transition probability $\{p_{x,y}\}_{y\in\mathcal{X}}$. Compared to classical methods [3] such as inverse of probability distribution function, rejection, or composition methods, the complexity of the computation of the next state is in $\mathcal{O}(1)$, and so does not depend on the problem size.

Consider a typical distribution $q = (q_0, \cdots, q_{k-1})$ on k states $\{y_0, \cdots, y_{k-1}\}$. The idea is to build a set of k thresholds $\{s_0, \cdots, s_{k-1}\}$ $0 \leqslant s_i \leqslant 1$, and k couples of states $\{(y_0, z_0), \cdots, (y_{n-1}, z_{n-1})\}$ with $y_i, z_i \in \mathcal{X}$, z_i is called the alias value of y_i. This construction should verify the following constraints :

$$\forall j \in \{0, \cdots, k-1\}, \quad q_j = \sum_{i=0}^{k-1} \left(s_i \mathbb{1}_{y_i = x_j} + (1 - s_i) \mathbb{1}_{z_i = x_j} \right). \tag{10}$$

Such a decomposition is built by a simple algorithm requiring $\mathcal{O}(k)$ steps. The implementation structure is described in [3]. From this structure the simulation runs as follows :

Algorithm 5 Aliasing generation

{ The values of s,y and z are preliminary stored in arrays of size k S,Y and Z.}
$u \leftarrow Random$;
$v \leftarrow Random$;
$i = int(u * k)$ { discrimination among k, int means the integer part }
if $v < S[i]$ **then**
  return $Y[i]$ { the standard value}
else
  return $Z[i]$ { the alias value}
end if

As an example consider the transitions from state 0, we get $(0.096, 0.226, 0.226, 0.226, 0.226)$ for neighbors $(0, 1, 2, 4, 7)$. The construction of the alias table is detailed in figure 6.

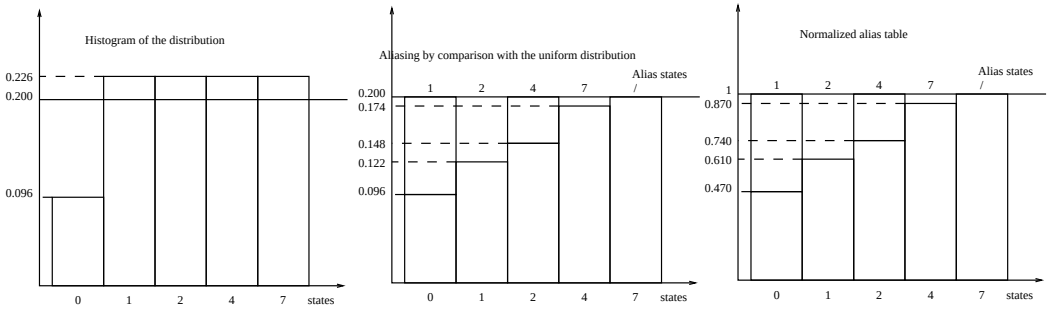


Fig. 6. Example of a computation of alias table

One should notice that this representation is not unique and, according remarks on the impact of representation on coupling time, we should use heuristics to build a "better" representation. In particular very interesting property of such a construction is that any permutation of two couples $(s_i, (y_i, z_i))$ and $(s_j, (y_j, z_j))$ provide

another random variable with exactly the same distribution. Moreover, if we replace some threshold s_i by $1 - s_i$ and exchange the values $Y[i]$ and $Z[i]$, the distribution of the result is also preserved.

Consequently, we have two steps in the computation of the simulation kernel. A first step compute for each state x the corresponding arrays S_x, Y_x, and Z_x (cf algorithm 5). The second step modify these arrays to guarantee termination in a finite number of steps. To simplify this step, we suppose that there exist some state x_0 such that the transition probability from x_0 to x_0 is strictly positive. This condition is stronger than aperiodicity and is generally verified in practical situations. If not, the matrix $\frac{1}{2}(Id + P)$ exhibits the same stationary distribution as P and could be used instead of P.

Because the Markov chain is irreducible there exist a spanning tree of the state space graph such that a path of positive probability exists in the tree from each state x or y to x_0. Because each state has an out-degree of 1 in the tree, it is always possible to place the next state of x in the tree (on the path to x_0) in the place $Y_x[0]$. Let $\alpha = \min_x \frac{S_x[0]}{d_x}$, with d_x the out-degree of state x in the graph. α is strictly positive and with probability α all the transitions occur on the arrows of the tree. Repeating this transition n times leads to a global coupling in state x_0. Now, we could apply corollary 1 and $\tau < +\infty$ almost surely. The algorithm terminates almost surely. Moreover, if we denote by D the depth of the tree, the coupling time is upper bounded by a geometric distribution with parameter α^D, probability that a burst of D sequential transitions occur on the tree.

The global algorithm of backward simulation is now :

From an implementation point of view, it appears that this structure could easily be adapted to the HBF format. We replace the values by the thresholds and add an array corresponding to the aliases. In our example we get the same line structure

	0	1	2	3	4	5	6	7	8	9	10	11
$Index\ of\ row$	0	5	9	13	17	21	25	29	33	37	41	45

The column index array remains, we compute the threshold array and the aliases :

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	···
$Column$	0	1	2	4	7	0	3	6	9	0	1	2	3	0	···
$Threshold$	0.470	0.610	0.740	0.870	1	1	0.712	0.808	0.904	0.003	0.161	0.161	1	0.323	···
$Column\ alias$	1	2	4	7	/	/	0	3	6	3	3	3	/	···	···

One should notice that the memory overhead of this representation is exactly the size of the column alias array. This is not too expensive with regards to the speedup offered by the alias computation. Typically the complexity of the generation of one step is in $\mathcal{O}(n)$ with n the total number of states.

In fact, the data structure is not really adapted to this algorithm, there is a lack of

Algorithm 6 Functional backward-coupling simulation with aliasing

for all $x \in \mathcal{X}$ **do**
$y(x) \leftarrow C(x)$ { choice of the **initial value** of the vector y, at $n = 0$, a trajectory finishing in state x has **cost** $C(x)$}
end for
repeat
u $\leftarrow$ Random; v $\leftarrow$ Random;{generation of u_{-n}}
for all $x \in \mathcal{X}$ **do**
{D_x is the out-degree of state x, S_x, Y_x and Z_x the arrays of size D_x, computed previously and corresponding to the alias method of the distribution $p_{x,.}$ }
$i = int(u * D_x)$
if v $< S_x[i]$ **then**
$y(x) \leftarrow y(Y_x[i])$;
else
$y(x) \leftarrow y(Z_x[i])$;
end if
end for
until All $y(x)$ are equal
return $y(x)$

locality in data and many cache miss and page defaults occur. Inversing the role of rows and columns avoid this problem. It will be discussed in a further paper.

4 Numerical results

4.1 Functionality tests

The aim of this first experiment is to validate the correctness of the algorithm and to illustrate the power of the aliasing method for full matrices. In that case, the matrix is stored by columns such that the aliasing method ensure data locality. In fact, it is of great interest when the state space is not too large such that the full matrix could be stored inside memory.

Because the objective was on the verification of the program, random matrices have been chosen. Coefficients are uniformly generated on $[0, 1]$ and we normalize the matrix to get a stochastic matrix. Then, it appears that the coefficients are all in the same order of magnitude and the aliasing technique is really efficient and coupling time is small.

Number of states	10	100	500	1000	3000
Mean coupling time	3.1	4.5	5.3	5.7	6.1
Mean execution time μs	3	17	170	360	1100

The mean execution time (excluding the precompiling of the table of aliases has been runned on a Pentium III 700MHz and 256Mb memory. Statistical estimations have been done on sample of size is 10000.

Problems appear when we increase the size of the matrix because of memory management and swap. These problems have been solved with the version of the software developed in the context of sparse matrices. It is interesting to note that we manipulate Markov chains with about 10 millions of transitions with a reasonable simulation time.

In this case the coupling efficiency is due to the structure of the matrix, the probability of coupling in one step is important. To explore the opposite situation, birth and death processes have also been implemented and simulated using various birth and death rates. We didn't take into account the sparsity of the matrix, alias table are then of the size of the state space. In this example we choose $\lambda = 0.7$ and $\mu = 1$, we are near the symmetric random walk.

Number of states	10	100	500	1000	3000
Mean coupling time	41	557	2850	5680	17000
Mean execution time μs	28	1800	88177	366000	3.5s

This is clearly the bad way to simulate these birth and death process, taking into account monotonicity [10] will improve greatly the simulation time.

4.2 Shared resources model

To validate the statistical properties of the simulator, a typical problem of shared resources has been analyzed. This model [5] is described by figure 7.

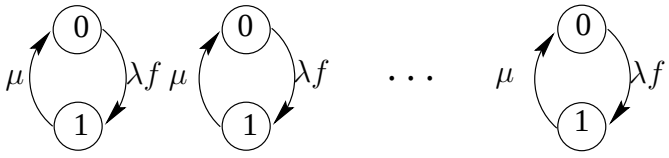


Fig. 7. Diagram of the shared resources model

In this example, N users share P resources. Each user could access at most one resource, it is then modeled by a two state automaton. The transition rate to get a resource is λ, the resource consumption rate is μ. In this system constraints between users appear when all

resources are occupied, then users should stay in state 0. This is modeled by the functional transition λf defined by

$$f(x_1, \cdots, x_n) = \begin{cases} 1 \text{ if } \sum_i x_i < P, \\ 0 \text{ if } \sum_i x_i = P. \end{cases}$$

where $(x_1, \cdots, x_N)$ is the state of the automata network.

The state space is included in $\{0,1\}^N$. Because at most P resources could be used the state space is of size

$$K = \sum_{k=0}^{P} C_N^k.$$

Varying parameters N and P leads to a family of models and we could fix the size of the model. Moreover, a simple analysis shows that such models are reversible and have a product form solution that could be computed separately. Then we are able to compare simulation results with analytical solutions. All of our χ^2 tests succeeded with a 95% confidence level.

To analyze the efficiency of functional coupling, we explore the marginal distribution of the number of occupied resources. The distribution of the coupling time is compared in the state coupling and functional coupling situation.

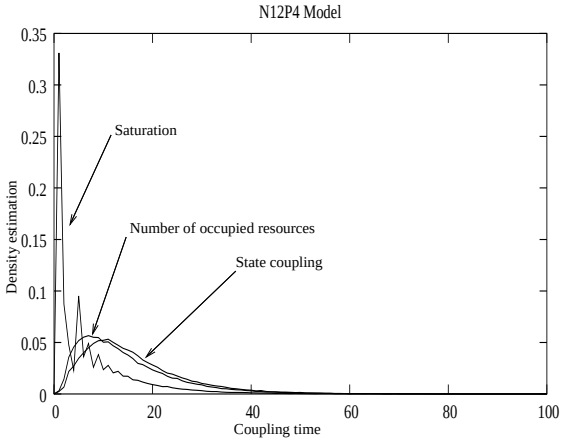


Fig. 8. Comparison on the model with $N = 12$ and $P = 4$ (sample size 10000)

On this curves, we show the density function of the coupling time for the generation of a typical state following the stationary distribution, the generation of a number of occupied resources and the generation of a $\{0,1\}$ variable indicating that the system is saturated. This illustrates clearly the efficiency of this method, the reduction of simulation time is about a factor of 10 on the mean coupling time.

4.3 Queuing network with blocking and priorities

In the context of networking, queuing networks models with finite capacities priorities are used to model access to networks and protocols that perform quality of service. In our model, (figure 9), we get N finite capacities queues, when front queues are full, arriving customers are lost. When the back queue is full, other servers are blocked.

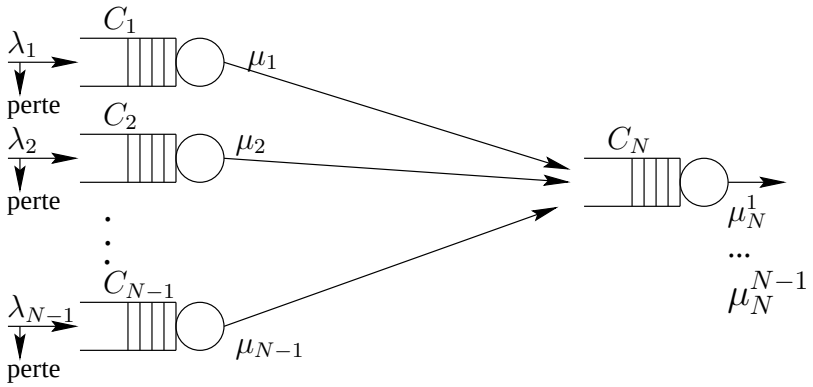


Fig. 9. Queueing network with rejection

The size of the state space is given by

$$K = \frac{1}{(N-1)!} \prod_{i=1}^{N-1} (C_i + 1) \times (C_{N-1} + i).$$

In the example we took 4 front queues with capacity 1 and the back queue with capacity 2. That leads to 80 states. If we study only the last queue we get only 8 states

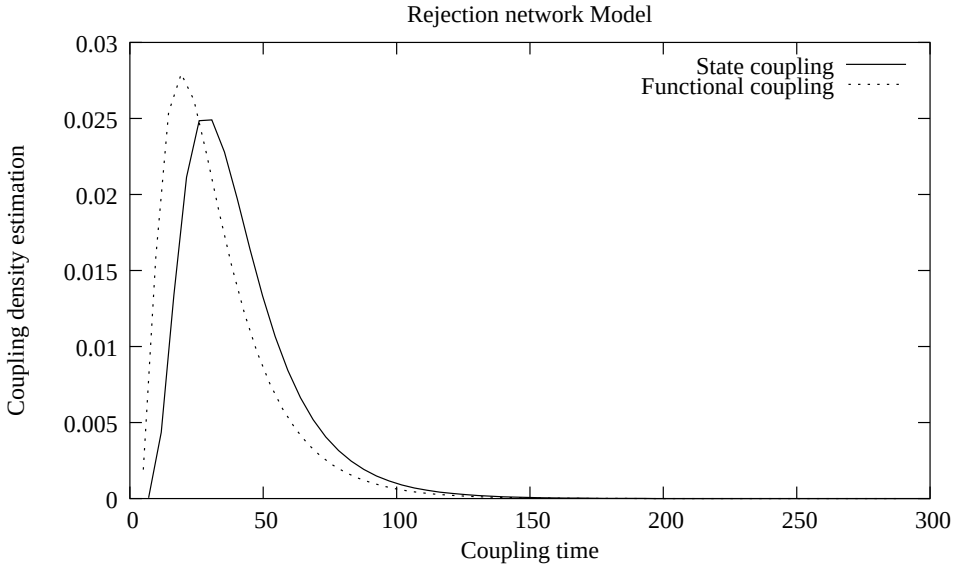


Fig. 10. State coupling and functional coupling

In the results representation (figure 10), even with a small number of states, the reduction is significant for the functional coupling.

4.4 Overflow queuing model

4.4.1 Model

To explore wide models with non analytical solution, we consider a set of K servers (figure 11). The system is fed by an input Poisson flow with rate λ. Each server has rate μ_i and the allocation policy is the following : any new customer is served by the first empty server; if all servers are occupied, the customer is rejected. This model is a particular case of Erlang model with priorities.

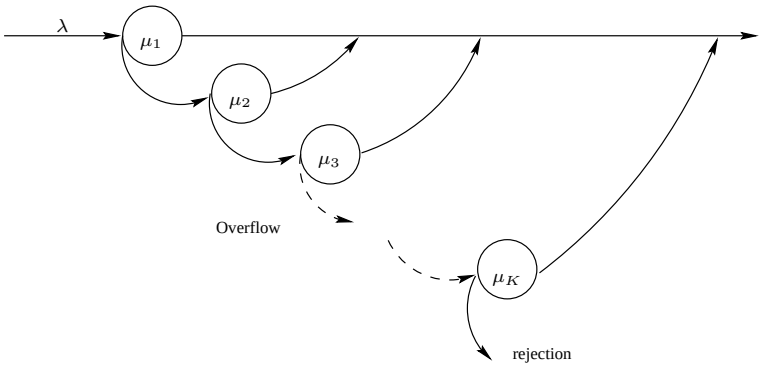


Fig. 11. A typical overflow model

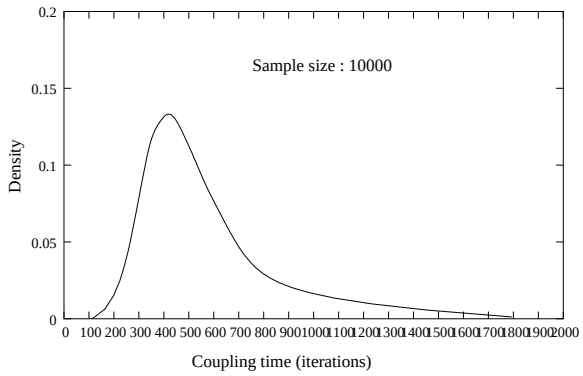
The state space is clearly isomorphic to $\{0,1\}^K$ with size $N = 2^K$, the number of non negative transitions is $m = 2N - 1 + \log_2 N.\frac{N}{2}$ and the maximum number of non negative elements per row is $K + 1$. The transition matrix is heavily sparse. It is important to notice that such a process is non-reversible and the associated matrix is non monotonic.

4.4.2 State coupling

In our case, the size of the model have been fixed to $K = 17$, but we also obtain results on models with size $K = 22$. The value of parameters are $\lambda = 2.6$ and $\mu_i = \frac{2}{i+1}$ (the global system is medium loaded). We simulate samples of size 10000.

We first compute samples of states and analyze the distribution of the coupling time (figure 12).

The statistical parameters of the distribution of coupling time are the following :



Parameter	Value
minimum	113
maximum	1794
1^{st} quartile	372
median	465
3^{rd} quartile	592
mean	498
variance	32306
Std	180

Fig. 12. Empirical distribution of the coupling time (overflow model)

First we notice that the distribution is quite regular with an exponential tail behavior. Moreover, the mean value 498 is small if we compare to the number of states $N = 2^{17} = 131072$. This is due at least in part to the coding of the transition function. Fortunately, this unimodal distribution is well centered and only 1% of the distribution exceeds two times the mean value.

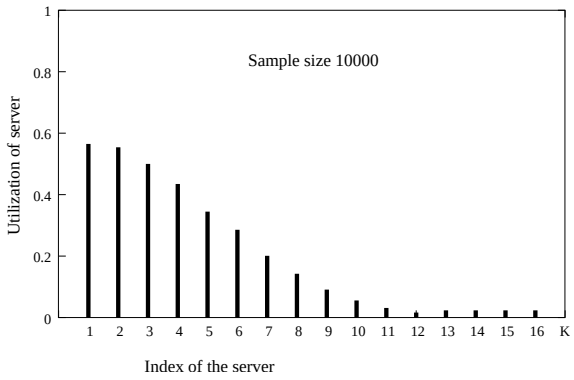
4.4.3 Functional coupling

Next, the functional version had been implemented to estimate directly the probability that a specific server is occupied, the distribution of the number of occupied servers and the probability that the system is not empty.

If we look at the number of occupied servers at steady state, we get the following histogram (figure 14).

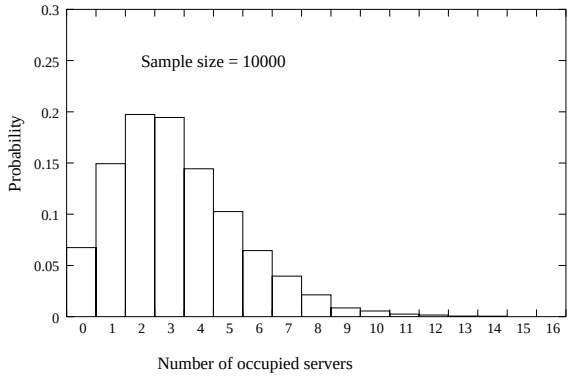
Finally, the utilization of the system had been computed, we got the probability that the system is empty is 0.067. One should notice that this functional is a simple restriction of the number of occupied servers at steady-state.

If we explore the coupling time for all of these functionals, we get the table given in the appendix page 24. The last line of this table indicates the statistics for the state coupling. These results are synthesized in a box-plot format on figure 16.



Each column of this diagram indicates the probability that the i^{th} server is occupied at steady-state. Because of the independance of generated values, according to the formula [8] the lenth of the confidence interval at 95% level is majorized by $\frac{\sigma 1.68}{\sqrt{10000}}$. In our case for a utilization factor in $[0,1]$ σ is bounded by $\frac{1}{2}$ and the error is less than 1%.
The figure 13 confirms that server occupation are in decreasing order, this is due the fact that the servers are not compensated by the slowness of servers with high index.

Fig. 13. Marginal distribution of the steady state vector (overflow model)



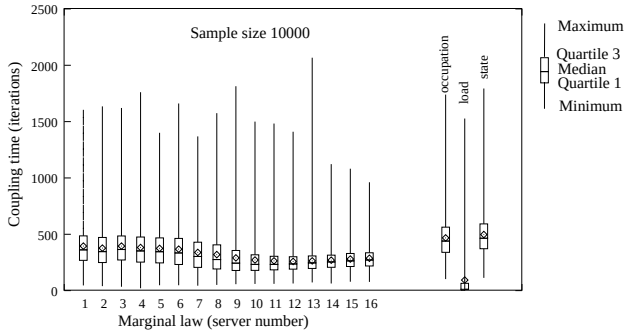
The distribution of the occupied server number at steady state shows that the system is never full, moreover, the probability that more than 14 is less than 10^{-4}. The mean load is 3.24 and the median 3.

Fig. 14. Number of occupied servers at steady state (overflow model)

5 Conclusion

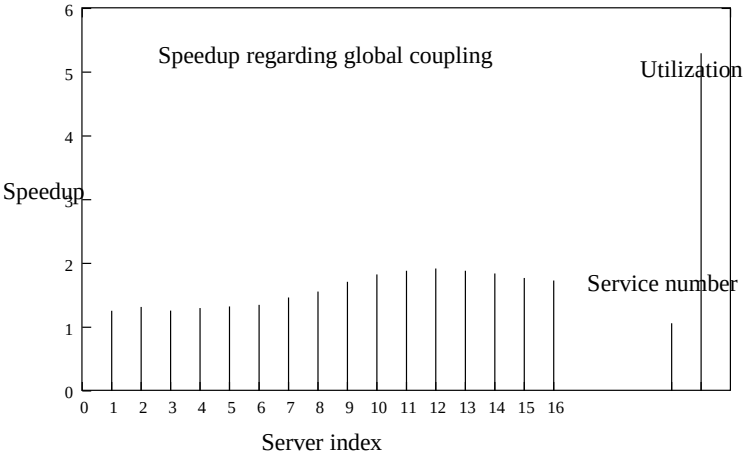
In this article, we adapt the methodology developed by Propp and Wilson [10] to discrete time Markov chain with finite state space. After some remarks on the representation of the Markov chain, we establish an algorithm that computes a transition function that ensure coupling almost surely.

Moreover, this method extends to the simulation of functionals of the steady states. It is shown on some example, that the coupling time is significantly decreased. Unfortunately, we are not able to estimate, from the representation of the Markov chain, the time reduction obtained with the functional.



It is clear that functional coupling is more efficient than state coupling. All distributions exhibit a regular shape with exponential decrease. According to confidence intervals, the comparison between the last marginal distribution coupling are not significant. One should notice that worst cases are in the same order of magnitude.

Fig. 15. Synthesis on functional coupling time (overflow model)



The respective gains are about 20% for the first marginals. But, in some cases, functional simulation runs two times quicker. The best result appears with the estimation of the global utilization (speedup of 5).

Fig. 16. Synthesis on functional coupling time (overflow model)

To test this method, a free software had been developed PSI (perfect simulator). From a standard description of the Markov chain (sparse representation) it implements the computation of the transition function. Then it generate forward samples, backward coupling and functional backward coupling.

To explore the scalability of the software, the overflow model have been tested with up to 22 servers. The state space is about 4 millions of states and, depending on parameters values, the sate coupling time is in the order of a 60 seconds. These promising first experiments suggest that this method could apply to larger state space. The limitation of the method is in memory. The memory needed for this model is about 600Mo.

This method is adapted to the general context of Markov chains. In particular, reversibility and monotonicity are not necessary. The limit of the method is then related to the complexity of storage of the transition matrix. We also need the storage of a vector which dimension is the size of the state space. Reduction of state storage appears when the model exhibits some regular structure (e.g. interacting system of particles, stochastic Petri net, queuing system,...). In that case, our methodology should be deeply modified to take into account the structure and adapt the representation. We look forward to further results in this direction.

References

[1] D. Aldous and A. Bandyopadhyay. How to combine fast heuristic markov chain monte carlo with slow exact sampling. *Electronic Commun. Probab.*, 6:79–89, 2001.

[2] A.A. Borovkov and S. Foss. Two ergodicity criteria for stochastically recursive sequences. *Acta Appl. Math.*, 34, 1994.

[3] P. Bratley, B.L. Fox, and L.E. Schrage. *A Guide to Simulation*. Springer-Verlag, 1983.

[4] P. Diaconis and D. Freedman. Iterated random functions. *SIAM Review*, 41(1):45–76, 1999.

[5] P. Fernandes. *Méthodes numériques pour la solution de systèmes markoviens à grand espace d'états*. PhD thesis, INPG, 1992.

[6] P. Fernandes, B. Plateau, and W. Stewart. Efficient descriptor vector multiplications in stochastic automata networks. *Journal of the ACM*, 45(3):381–414, 1994.

[7] O. Haggstrom. *Finite markov chains and algorithmic applications*. Cambridge University Press, 2002.

[8] R. Jain. *The Art of Computer Systems Performance Analysis Techniques for Experimental Design, Measurement, Simulation, and Modeling*. John Wiley & Sons, 1991.

[9] R.M. Loynes. The stability of queues with non independent inter-arrival and service times. *Proc. Cambridge Ph. Soc.*, 58:497–520, 1962.

[10] J.G. Propp and D.B. Wilson. Exact sampling with coupled markov chains and applications to statistical mechanics. *Random structures and Algorithms*, 9:223–252, 1996.

[11] O Stenflo. *Ergodic theorems fory Iterated Function Systems controlled by stochastic sequences*. Doctoral thesis n° 14, Umea university, 1998.

[12] O Stenflo. Ergodic theorems for markov chains represented by iterated function systems. *Bull. Polish Acad. Sci. Math*, 2001.

[13] W.J. Stewart. *Introduction to the Numerical analysis of Markov Chains*. Princeton, 1994.

[14] A.J. Walker. An efficient method for generating discrete random variables with general distributions. *ACM Trans. Math. Software*, 3:253–256, 1974.

Statistics on the overflow model

Functional	Min	Max	1^{st} quartile	Median	3^{rd} quartile	Mean	Variance	Std
Server 1	46	1608	269	362	486	395.5	32157.5	179.3
Server 2	40	1635	249	346	472	377.7	32321.3	179.8
Server 3	34	1621	272	365	485	395.0	30714.7	175.3
Server 4	21	1761	253	352	476	382.8	33943.3	184.2
Server 5	48	1401	246	345	468	375.4	31998.4	178.9
Server 6	48	1661	232	334	463	368.3	34928.0	186.9
Server 7	44	1369	206	303	430	339.3	32380.9	179.9
Server 8	51	1575	192	276	406	319.5	29815.5	172.7
Server 9	57	1814	179	244	356	290.6	25998.2	161.2
Server 10	59	1500	180	233	318	272.2	20464.4	143.1
Server 11	60	1483	184	233	305	264.2	16036.1	126.6
Server 12	63	1410	191	237	301	258.9	11450.5	107.0
Server 13	73	2067	197	246	308	263.9	9630.5	98.1
Server 14	63	1123	207	255	315	270.2	8665.3	93.1
Server 15	79	1082	214	264	330	280.8	9249.3	96.2
Server 16	77	962	219	272	336	287.3	9407.7	97.0
Server number	102	1740	340	440	563	468.2	32521.5	180.3
System utilization	1	1527	4	12	65	94.0	29762.5	172.5
State	113	1794	372	465	592	497.7	32306.8	179.7