Perfect Simulation of Stochastic Automata Networks[⋆]

Paulo Fernandes[1], Jean-Marc Vincent[2], and Thais Webber[3⋆⋆]

[1] PUCRS–CNPq `Paulo.Fernandes@pucrs.br`
[2] LIG–MESCAL `Jean-Marc.Vincent@imag.fr`
[3] PUCRS–CAPES `twebber@inf.pucrs.br`

Abstract. The solution of continuous and discrete-time Markovian models is still challenging mainly when we model large complex systems, for example, to obtain performance indexes of parallel and distributed systems. However, iterative numerical algorithms, even well-fitted to a multidimensional structured representation of Markov chains, still face the state space explosion problem. Discrete-event simulations can estimate the stationary distribution based on long run trajectories and are also alternative methods to estimate performance indexes of models. Perfect simulation algorithms directly build steady-state samples avoiding the warm-up period and the initial state bias of forward simulations. This paper introduces the concepts of backward coupling and the advantages of monotonicity properties and component-wise characteristics to simulate Stochastic Automata Networks (SAN). The main contribution is a novel technique to solve SAN descriptions originally unsolvable by iterative methods due to large state spaces. This method is extremely efficient when the state space is large and the model has dynamic monotonicity because it is possible to contract the reachable state space in a smaller set of maximal states. Component-wise characteristics also contribute to the state space reduction extracting extremal states of the model underlying chain. The efficiency of this technique applied to sample generation using perfect simulation is compared to the overall efficiency of using an iterative numerical method to predict performance indexes of SAN models.

1 Introduction

The solution of Discrete and Continuous-Time Markov Chains (MC) [1] is still challenging mainly when we model large complex systems, such as parallel and distributed systems. The size of the infinitesimal generator to be stored has limits and also the available numerical algorithms must deal with more huge and complex representations. The steady-state is given by the long-run probability distribution obtained by the solution of the linear system $\boldsymbol{\pi} \mathcal{Q} = 0$, where $\boldsymbol{\pi}$ is the probability vector of size n which is initially distributed as $\boldsymbol{\pi}_0$ where $\boldsymbol{\pi}_0 \geq 0$ and $\sum_{i=0}^{n} \boldsymbol{\pi}_i = 1$. However, even with algorithms well-fitted to a multidimensional structured representation of MC, the state space explosion is still a problem when solving models.

Stochastic Automata Networks (SAN) is considered a high-level formalism [2] to represent structured MC. The available numerical solutions take advantage of all structural information in the original model description to obtain a compact format to store

[⋆] The order of the authors is merely alphabetical.
[⋆⋆] Corresponding author.

and manipulate the descriptor numerically [3–5]. One of the major problems with structured representations is the insertion of unreachable states in the product state space, but to cope with that there are very efficient approaches to generate the reachable set [6, 7]. It remains an open problem the efficient solution of large and complex models where all, or almost all, states are reachable.

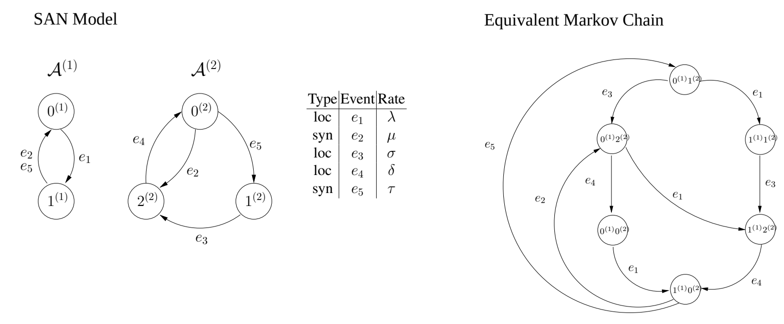
Simulation approaches are alternative methods to estimate indexes of performance models when the numerical solution is no longer sufficient. Based on discrete-event simulation or on Markov properties, simulations estimate the stationary distribution $\boldsymbol{\pi}$ based on long run trajectories. The first approaches to simulate a SAN, or any other structured formalism, focus on the concept of events [8]. Such event-driven dynamics implements a hierarchy of events inside the automata structure starting from a predefined initial global state. Despite of this event-driven choice, the problem of how long one have to run the simulation, *i.e.*, the *burn-in time* period, still remains open in forward simulations [9]. The system is simulated until it is considered that reached the stationary regime. After this time, the simulation is no more dependent of the initial state chosen due to the stationary assumption.

Propp and Wilson [10] proposed a backward coupling simulation method where the problem of biased samples is completely solved. Perfect simulation algorithms directly build steady-state samples avoiding the warm-up period and the initial state bias. The method proposes the running of trajectories in parallel, starting from all possible states, and their coupling guarantees the samples confidence. This method is extremely efficient when the state space is large and the model has dynamic monotonicity because this will determine the number of trajectories in parallel needed to run.

This paper introduces the concepts of backward coupling and the advantages of monotonicity properties to simulate SAN models. The structural information in the original SAN description can be used to contract even more the state space, analysing component-wise characteristics for example. The main contribution is the adaptation of a new simulation technique to SAN models originally unsolvable by iterative methods due state space explosion. Monotone backward coupling methods can run with a reduced state space since models have a monotonic behavior. The efficiency of sample generation using perfect simulation is compared to the overall efficiency of using an iterative numerical method to predict performance indexes of SAN models.

2 Stochastic Automata Networks

The Stochastic Automata Networks formalism (SAN) is an analytical method to obtain performance indexes of systems. It is proposed by Plateau [2] and its basic idea is to represent a whole system by a collection of K subsystems or chains described as K stochastic automata $\mathcal{A}^{(k)}$, with $k \in [1..K]$. In each of these automata the transitions among states are labeled by events. Each event includes probabilistic and timing information, and the network of automata has a set ξ of all possible events in the model. This framework defines a modular way to describe continuous and discrete-time Markovian models [11]. The SAN formalism has exactly the same application scope as the Markov Chain formalism [12, 1].



Type	Event	Rate
loc	e_1	λ
syn	e_2	μ
loc	e_3	σ
loc	e_4	δ
syn	e_5	τ

Fig. 1. Example of a SAN model and equivalent MC

Definition. Each automaton $\mathcal{A}^{(k)}$ has a set $\delta^{(k)}$ of local states $s^{(i)}$ where $i \in \{1 \dots n_k\}$, interconnected by transitions and their respective events. The constant n_k is the cardinality of $\delta^{(k)}$, *i.e.*, the total number of states in automaton $\mathcal{A}^{(k)}$.

Definition. A global state $\tilde{s}$ of a SAN model with K automata is a vector $\tilde{s} = \{s^{(1)}; \dots; s^{(K)}\}$ where each automaton $\mathcal{A}^{(k)}$ is in the local state $s^{(k)} \in \delta^{(k)}$.

Definition. The set of all global states is called *product state space*. The product state space $\mathcal{X}$ of a SAN model is the Cartesian product of all sets $\delta^{(k)}$.

Considering the product state space $\mathcal{X}$, the system is composed by a set of global states as $\tilde{s}$ and also a finite collection $\xi = \{e_1, \dots, e_P\}$ of P events. Since models with discrete state space can also be described as discrete-event systems [13], the set ξ can be defined with an associated transition function Φ between global states.

Definition. The transition function defined by $\Phi(\tilde{s}, e_p) = \tilde{r}$ ($p \in [1..P]$) is the set of rules that associate to each global state $\tilde{s} \in \mathcal{X}$ a new global state denoted by $\tilde{r} \in \mathcal{X}$, through the firing of the transition labeled by event $e_p \in \xi$.

In each global state $\tilde{s}$ some events are enabled, *i.e.*, they change the global state $\tilde{s}$ into another state $\tilde{r}$. However, not all events may occur from a given global state. In those cases the transition function assigns the permanence in the same global state.

Definition. An event e_p is said to be enabled in the global state $\tilde{s} \in \mathcal{X}$, iff $\Phi(\tilde{s}, e_p) = \tilde{r}$, and $\tilde{s} \neq \tilde{r}$, and $\tilde{r} \in \mathcal{X}$. Analogously, an event is said to be disabled in state $\tilde{s}$, iff $\Phi(\tilde{s}, e_p) = \tilde{r}$, and $\tilde{s} = \tilde{r}$.

The SAN model construction as a Markov process has the rates of each event e_p seen as intensities λ_p of Poisson processes, and they are supposed to be independent.

The SAN description has a table of events extracted from ξ and uniformization techniques are used to introduce the independence between these events. The uniformized process is driven by the Poisson process with rate $\Lambda = \sum_{p=1}^{P} \lambda_p$ and generates at each time an event $e_p \in \xi$ according to the distribution $\left(\frac{\lambda_1}{\Lambda}, \dots, \frac{\lambda_p}{\Lambda}\right)$.

Definition. The dynamic of the system is defined by one initial global state $\tilde{s}_0 \in \mathcal{X}$ and a sequence of events $e = \{e_p\}_{p \in \mathcal{N}}$. The sequence of states $\{\tilde{s}_n\}_{n \in \mathcal{N}}$ is a stochastic recursive sequence typically given by: $\tilde{s}_{n+1} = \Phi(\tilde{s}_n, e_{p+1})$ for $p \geq 0$ and is called a *trajectory*.

$\tilde{s} \in \mathcal{X}^{\mathcal{R}}$	$\tilde{r} = \Phi(\tilde{s}, e_p), e_p \in \xi$				
	$\Phi(\tilde{s}, e_1)$	$\Phi(\tilde{s}, e_2)$	$\Phi(\tilde{s}, e_3)$	$\Phi(\tilde{s}, e_4)$	$\Phi(\tilde{s}, e_5)$
{0;0}	**{1;0}**	{0;0}	{0;0}	{0;0}	{0;0}
{0;1}	**{1;1}**	{0;1}	**{0;2}**	{0;1}	{0;1}
{0;2}	**{1;2}**	{0;2}	{0;2}	**{0;0}**	{0;2}
{1;0}	{1;0}	**{0,2}**	{1;0}	{1;0}	**{0;1}**
{1;1}	{1;1}	{1;1}	**{1;2}**	{1;1}	{1;1}
{1;2}	{1;2}	{1;2}	{1;2}	**{1;0}**	{1;2}

Table 1. Application of $\Phi(\tilde{s}, e_p)$ for the model of Fig. 1

The global process execution [14, 15] described is related to the underlying uniformized Markov chain. Its transitions are given by Φ applications over $\mathcal{X}$. However, it is common to have global states that are not reachable by any other global state through a transition. Due to this SAN models have established a reachable state space, *i.e.*, the set of global states $\tilde{s} \in \mathcal{X}$ that composes the related MC. The others are considered unreachable global states in the model.

Definition. The reachable state space $\mathcal{X}^R_{\tilde{s}_0}$ (or $\mathcal{X}^R$) is an irreducible component obtained from a given initial global state $\tilde{s}_0 \in \mathcal{X}$ and successive firing of events in ξ. Each global state $\tilde{s}$ reached by any possible combination of events is included in this set.

Note that SAN descriptions must have only one Markovian generator [11], the associated Markov chain contains a set of all global states $\tilde{s} \in \mathcal{X}^R$ that certainly can be reached through the firing of any event. Figure 1 is a SAN model with two automata $A^{(i)}$, and five events ($|\xi| = 5$), and their constant rates represented here by greek letters. The equivalent MC represents the reachable state space $\mathcal{X}^R$ of the model which is a subset of $\mathcal{X}$ ($\mathcal{X}^R \subseteq \mathcal{X}$).

A simple procedure to find reachable states is to apply the notion of stochastic recursive transition function mainly when the reachability function is not explicit in the SAN formal descriptions[4]. Table 1 shows the transition function application for the SAN example in Figure 1, considering all global states $\tilde{s} \in \mathcal{X}^R$ and all events $e_p \in \xi$. The resulting global states $\tilde{r} = \Phi(\tilde{s}, e_p)$ are represented, being those corresponding to possible transitions marked in bold face, *i.e.*, those corresponding to enabled events[5].

Definition. A SAN model is called *well-formed* iff the $\mathcal{X}^R$ component is unique and irreducible.

[4] SAN descriptions can define the $\mathcal{X}^R$ set through the insertion of a reachability function. The boolean evaluation of this function, when applied to every global state inside $\mathcal{X}$, returns the reachable states in $\mathcal{X}^R$. More details can be found in [16, 17].

[5] It is important to observe that the transition function Φ is a theoretical definition that is not necessarily used in current SAN solvers implementation. However, algorithms can be implemented to take advantage of transition functions identifying also the reachability set $\mathcal{X}^R$.

3 SAN Backward coupling simulation

The first approaches to simulate SAN models focused on the concept of transitions and events, instead of having a focus on state transition matrices, *i.e.*, the descriptor [8]. Such event-driven dynamics implements a hierarchy of events inside the automata structure starting from a pre-defined initial global state. Despite of this event-driven choice, the problem of how long one should run the simulation still remains open using forward simulation approaches [9]. Moreover, simulation techniques use the *Random* function to establish the activation of an event inside ξ, considering the current global state analysed, then leading to the next global state inside $\mathcal{X}^{\mathcal{R}}$ going forward in time. The system is often simulated until it reaches its stationary regime. The duration of this step is called the *burn-in time* of the simulation and it determines that the process is no more dependent of the initial state chosen, due the stationary assumption. Since the major challenge in these techniques is to fix a *burn-in time* to allow collecting samples, the Perfect simulation technique enables to compute samples exactly distributed according to the stationary distribution of the Markov process. Propp and Wilson [10] proposed a scheme based on backward coupling, *i.e.*, the *Coupling from the Past* (CFTP) method. The problem of fixing initial state present in forward techniques is completely solved since the proposed idea is to start trajectories in parallel from all possible states.

Algorithm 1 SAN Backward coupling simulation

```
1: for all $\tilde{s} \in \mathcal{X}^R$ do
2:     $\omega(\tilde{s}) \leftarrow \tilde{s}$ { choice of the initial value of vector $\omega$}
3: end for
4: repeat
5:     e ← Generate-event( ) { generation of $e$ according the distribution $(\frac{\lambda_1}{\Lambda} \dots \frac{\lambda_{\mathcal{E}}}{\Lambda})$}
6:     $\tilde{\omega} \leftarrow \omega$ { copying vector $\omega$ to $\tilde{\omega}$}
7:     for all $\tilde{s} \in \mathcal{X}^R$ do
8:         { computing $\omega(\tilde{s})$ at time 0 of trajectory issued from $\tilde{s}$ at time $-\tau^*$}
9:         $\omega(\tilde{s}) \leftarrow \tilde{\omega}(\Phi(\tilde{s}, e))$
10:    end for
11: until All $\omega(\tilde{s})$ are equal
12: Return $\omega(\tilde{s})$
```

The coupling of trajectories guarantees the generation of unbiased samples and it ended the *burn-in time* problem. The number of steps (or events applied) to couple all trajectories we denote *coupling time* τ. Figure 2 illustrates the backward coupling, all trajectories issued from all global states of the SAN example (Figure 1) at time -8 coupled in a state at time 0. Since the coupling time τ^* of the backward scheme is almost surely finite, the scheme provides a sample distributed according the steady-state distribution. Given the set of reachable states $\mathcal{X}^R$, a set $\mathcal{E}$ of randomly generated events and the transition function $\Phi : \mathcal{X}^R \times \mathcal{E} \rightarrow \mathcal{X}^R$: issuing from all global states of $\mathcal{X}^R$, going backward in time, the set of trajectories will couple for a given sequence of events $\{e_n\}_{n\in\mathcal{N}}$ at time 0, *i.e.*, $|\Phi(\mathcal{X}^R, \{e_n\}_{n\in\mathcal{N}})| = 1$.

SAN models have an underlying Markov chain so perfect simulation principles can be applied to obtain the global states probabilities in the stationary distribution. For perfect simulation execution, it is mandatory a *well-formed* SAN description, *i.e.*, the model must produce valid global states as input for the simulation algorithm. For backward simulations the set of trajectories running in parallel can be at least the $\mathcal{X}^R$ set, when of course the $\mathcal{X}^R$ set is an unordered set of global states. Algorithm 1 initializes the vector ω with all global states $\tilde{s} \in \mathcal{X}^R$ at simulation time $-\tau^*$, supposing a well-formed SAN model. At each simulation iteration one event is generated through the call of a *Random* function and the related transition functions are applied to each position of ω. Each new state generated indexes the vector $\tilde{\omega}$ which has the last version of ω stored. This process is called backward coupling because we compute $\omega(\tilde{s})$ at time 0 of trajectory issued from $\tilde{s}$ at time $-\tau^*$. This procedure will be repeated until all positions of vector ω have the same resulting state $\tilde{s}$, *i.e.*, all trajectories running in parallel have coupled. The sample of each iteration is then collected for statistical analysis.

3.1 Monotonicity Properties

The size of $\mathcal{X}^R$ can be exponential in the size of the model and it can be difficult to generate and really huge to deal, so it becomes a limitation for backward coupling methods. As pointed out in Propp and Wilson [10], CFTP methods are much easier to implement when the state space $\mathcal{X}$ is ordered and the underlying Markov chain has the monotonicity property. A known partial order of $\mathcal{X}$ is favorable to the use of monotonic functions since it allows the identification of maximal states and a considerable coupling time reduction. Models with an underlying Markov chain having this property can be optimized to run a monotone backward coupling procedure with less initial states.

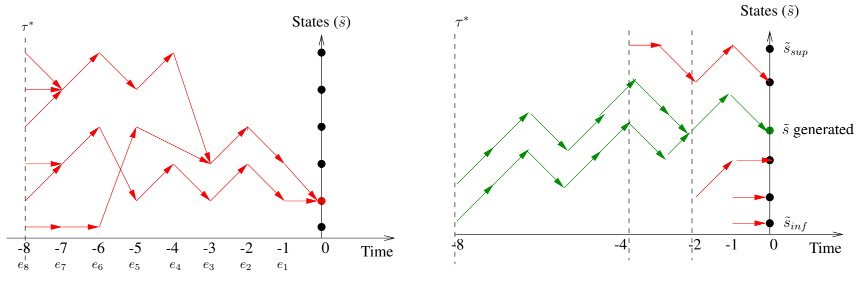


Fig. 2. Illustration of Backward and Monotone Backward coupling

Definition. An event $e_p \in \xi$ is said to be monotone if it preserves the partial ordering ($<$ order) on $\mathcal{X}$. That is $\forall(\tilde{s}, \tilde{s}') \in \mathcal{X} \quad \tilde{s} < \tilde{s}' \implies \Phi(\tilde{s}, e_p) < \Phi(\tilde{s}', e_p)$.
If all events are monotone, the global system is said to be monotone.

The monotonicity property of events guarantees the existence of a set of maximal states $\mathcal{X}^{\max}$ and a set of minimal states $\mathcal{X}^{\min}$. These sets are composed of states which there is no greater (or lower) state than itself in the chain. So the transitions fired from maximal states do not create states greater than these ones (or transitions fired from minimal states do not create states lower than minimal ones).

Definition. Suppose the global states $\tilde{s}_1, \tilde{s}_2 \in \mathcal{X}$, a state $\tilde{s}_1$ is minimal if there exists a state $\tilde{s}_2$ such that $\tilde{s}_2 \leq \tilde{s}_1$ then $\tilde{s}_2 = \tilde{s}_1$. Then $\tilde{s}_1 \in \mathcal{X}^{\min}$. Analogously, given states $\tilde{s}_3, \tilde{s}_4 \in \mathcal{X}$, a state $\tilde{s}_3$ is maximal if there exists a state $\tilde{s}_4$ such that $\tilde{s}_4 \geq \tilde{s}_3$ then $\tilde{s}_4 = \tilde{s}_3$. Then $\tilde{s}_3 \in \mathcal{X}^{\max}$.

Algorithm 2 SAN Monotone backward coupling simulation

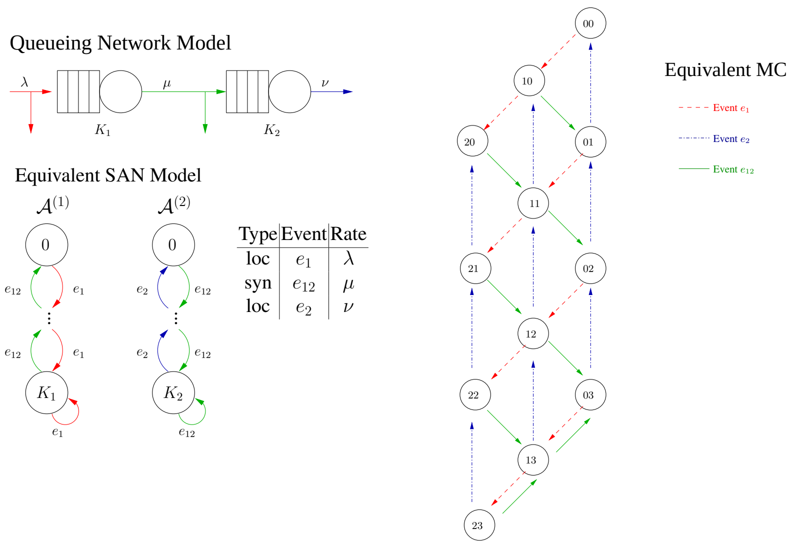
```
1: $n = 1$
2: $E[1] \leftarrow$ Generate-event( ) { array $E$ stores the backward sequence of events}
3: repeat
4:     $n \leftarrow 2n$ {doubling scheme}
5:     for each $\tilde{s} \in \mathcal{X}^M$ do
6:         $\omega(\tilde{s}) \leftarrow \tilde{s}$ { initial states at time $-n$}
7:     end for
8:     for $i = n$ downto $(\frac{n}{2} + 1)$ do
9:         $E[i] \leftarrow$ Generate-event( ) { generate events from $(-\frac{n}{2} + 1)$ to $-n$, events from $-1$ to
           $(-\frac{n}{2} + 1)$ have been generated in a previous loop}
10:    end for
11:    for $i = n$ downto $1$ do
12:        for each $\tilde{s} \in \mathcal{X}^M$ do
13:            $\omega(\tilde{s}) \leftarrow \Phi(\omega(\tilde{s}), E[i])$ { $\omega(\tilde{s})$ is the state at time $(-i - 1)$ of the trajectories issued
               from $\tilde{s}$ at time $-n$}
14:        end for
15:    end for
16: until All $\omega(\tilde{s})$ are equal
17: Return $\omega(\tilde{s})$
```

Definition. Suppose a given partial order of $\mathcal{X}$ and consequently a maximal set $\mathcal{X}^M$, if all trajectories issued from $\mathcal{X}^M$ coupled at time 0, then they will also coalesce for all states in $\mathcal{X}^R$.

Since $\mathcal{X}$ is finite and the events are monotone, the number of trajectories in parallel can be reduced running simulation only over the $\mathcal{X}^M = \mathcal{X}^{\max} \cup \mathcal{X}^{\min}$ set. Starting trajectories and going from the past from $\mathcal{X}^M$ maximal global states, when all trajectories collapsed, we also obtain a sample of the stationary regime. Figure 2 illustrates the monotone backward coupling, where there are only one *infimum* state $\tilde{s}_{inf}$ and one *supremum* state $\tilde{s}_{sup}$ in the state space $\mathcal{X}^R$. All trajectories issued from $\tilde{s}_{inf}$ and $\tilde{s}_{sup}$ are computed from time -2^k to 0 until trajectories collapsed at time 0. If we assume $\mathcal{X}^R$ as a *lattice*, then every state $\tilde{s}_i$ is between the state $\tilde{s}_{inf}$ and the state $\tilde{s}_{sup}$, $\tilde{s}_{inf} \leq \tilde{s}_i \leq \tilde{s}_{sup}$. Considering the SAN context, if we know the global states $\tilde{s}_{inf}$ and $\tilde{s}_{sup}$ (or a $\mathcal{X}$ subset of maximal states, *i.e.*, $\mathcal{X}^M$) we can run a monotone version. It uses a coupling vector ω of $|\mathcal{X}^M|$ positions running these trajectories in parallel. Also, it needs to store the events generated of the whole trajectory, because it uses a *doubling scheme* structure [10] to generate and apply events in each trajectory. At each step in the past, the *coupling time* τ^* needed (*i.e.*, the length of the step) is multiplied by 2 (Algorithm 2, line 4).

Canonical component-wise ordering in SAN Many models are naturally ordered as markovian queueing networks [18–20] due the natural order on integer. The partial order of the product state space can be established using for example component-wise ordering concepts. SAN descriptions derived from monotone queueing networks can be simulated taking advantage of having only the canonical minimum (all queues empty) and maximum (all queues full) states. Then only two paths need to simulate in parallel with the monotone algorithm version.

The canonical component-wise ordering means that the underlying Markov chain structure of the model can be viewed as a *lattice*, *i.e.*, all global states have the same *supremum* and *infimum* states. Given two arbitrary global states $\tilde{s}_1, \tilde{s}_2 \in \mathcal{X}$, and verifying $\tilde{s}_1 \leq \tilde{s}_2$, it will often possible to say which is the largest state [21]. The extremal states are given by the first and the last state of $\mathcal{X}$ considering it is ordered lexicographically. The complexity to solve these families of models is then constant (two trajectories in parallel) and the simulation is no more limited by the size of $\mathcal{X}$ but only by τ.



Type	Event	Rate
loc	e_1	λ
syn	e_{12}	μ
loc	e_2	ν

Fig. 3. QN conversion to a SAN model and the ordered $\mathcal{X}^R$.

Supposing the queueing system of two queues in Figure 3 with capacities K_1 and K_2 respectively. The $\mathcal{X}$ size of this network is given by the Cartesian product $(K_1+1) \times (K_2+1)$ and all global states $\tilde{s} \in \mathcal{X}^R$ (equivalent MC) are reachable ($\mathcal{X}^R \cong \mathcal{X}$). The

SAN model has two automata $A^{(1)}$ and $A^{(2)}$ respectively, and three events $e_p \in \xi$ (since two are local events and one is a synchronizing event in the model) with their rates. The events e_1, e_{12} and e_2 are monotone according canonical component-wise ordering of $\mathcal{X}$, *i.e.*, there is no event in the model changing the partial order of states in $\mathcal{X}$. The application of the transition function $\Phi(\tilde{s}, e_p)$, for each event $e_p \in \xi$, considering each state $\tilde{s} \in \mathcal{X}^R$, is dependant of the min and max functions[6] evaluations for each global state (in this example the global state to be evaluated has only two local states to observe $\tilde{s} = \{s_1; s_2\}$).

The minimum and maximum global states are extracted from the underlying Markov chain, but they consider the minimal and maximal local states of each automaton $\mathcal{A}^{(k)}$ defined by natural order on integer. Supposing $K_1 = 2$ and $K_2 = 3$, the maximal set can be considered $\mathcal{X}^M$ =$\{\{0;0\}$,$\{2;3\}\}$. The minimal local state of both automata is the state 0, and the maximal local state is 2 for automaton $A^{(1)}$, and 3 for automaton $A^{(2)}$ respectively. The simulation could run only two trajectories in parallel: all queues empty (minimal local states $\{0;0\}$) and all queues full (maximal local states $\{K_1;K_2\}$). The assumption of existing one minimum and one maximum local state per automaton which guarantees the exact sampling, can be applied also for huge models following component-wise principle.

Non-lattice component-wise ordering in SAN Glasserman and Yao [13] investigated the search for partial (and total) ordering in discrete-event models looking at their own structure, naturally retaining the order in which states in the chain are accessed firing the respective events. This procedure incrementally generates a *feasible set*, until all states are accessed (total ordering), or a given partial ordering is identified. So we can consider the feasible set as an ordered representation of the $\mathcal{X}^R$ set. However, in the absence of a canonical component-wise model formation, for each event in the model, the state space ordering must be constructed firing events in the underlying chain structure. If we have the same subchains ordering for the events, this means that exists a partial order for $\mathcal{X}^R \subseteq \mathcal{X}$ $(<)$, when it is possible to compare two states for a given event $e_p \in \mathcal{E}$, independent of event rates.

The ordering construction for the queueing system example when already exists a canonical formation leads us to a *lattice* where there are two maximal global states as seen in Figure 3. But without this characteristic the search for a order could be really unfeasible for huge models and with a high enhanced computational cost. The global states ordering in $\mathcal{X}$ becomes not relevant if we can extract through the transition function applications a smaller set of extremal global states of the Markov chain, *i.e.*, not retaining the order of access of states but verifying if the next state in a transition is greater than the current state. This means that if we walk in the chain applying the transition function successively we can reach and collect the extremal elements (Algorithm 3).

The component-wise ordering supposes that local states have a predefined order, then the Cartesian product of states generates automatically ordered global states. The states will always have transitions to greater or lower global states indexes. So when

[6] $x \wedge y = min(x, y)$ and $x \vee y = max(x, y)$.

Algorithm 3 SAN extremal states identification in component-wise models

```
1:  for each $\tilde{s} \in \mathcal{X}^R$ do
2:      max ← true;
3:      for each $e_p \in \xi$ do
4:          st ← $\Phi(\tilde{s}, e_p)$; { firing transition }
5:          if (st $\geq \tilde{s}$)
6:              max ← false; break;
7:      end for
8:      if (max = true)
9:          Add state $\tilde{s}$ in $\mathcal{X}^M$; { array $\mathcal{X}^M$ stores the extremal states identified}
10: end for
11: Return array $\mathcal{X}^M$;
```

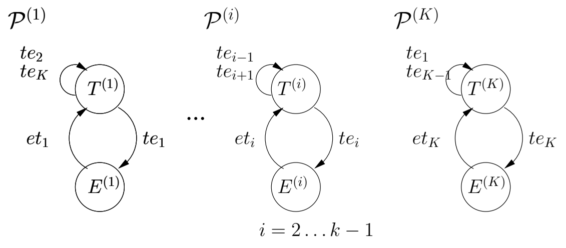
there is no possible transition to be fired to a greater state, this means we find an extremal state in the chain. In this case, models can have more than two maximal states (according to the $\mathcal{X}^R$ set analysis), *i.e.*, $\mathcal{X}^M$ is now the set of extremal elements formed by global states where the successive transition function application stops when it does not return greater states. Doing this, is not mandatory to know the state space partial ordering but only to identify the subset $\mathcal{X}^M$ of extremal states to run the monotone backward coupling algorithm. The complexity to find extremal global states is given by $|\mathcal{X}^R| \times |\xi|$. The computational cost to run perfect simulation is now dependent of the number of initial global states inside the set $\mathcal{X}^M$.

4 Resource Sharing with Mutual Exclusion

Our case study is the classical model of resource sharing with mutual exclusion and some variations. In this section we show the product state space contraction regarding natural structured formation of these models, and also issues related to the exploitation of monotonicity properties for perfect simulation. All simulation examples were executed on a PC architecture with a 3.2 GHz Intel Xeon processor under Linux operating system, with 1 GByte of memory. The execution times presented consider only usertime estimation running *PEPS* software tool and the perfect simulation module developed (Perfect *PEPS*), *i.e.*, they do not take account of other users in the machine or operating system execution.

4.1 Dining Philosophers without Reservation

The dining philosophers problem is summarized as K philosophers sitting at a table doing one of two things - eating or thinking. The philosophers sit at a circular table. with a large bowl of food in the center. A fork F_k is placed between each philosopher P_k, and as such, each philosopher has one fork to his left and one fork to his right. The philosopher must have two forks (at the same time) to eat. Figure 4 shows the correspondent SAN model that has K automata $P^{(k)}$ representing the philosophers, each one with two states: $T^{(k)}$ (thinking) and $E^{(k)}$ (eating). The product state space $\mathcal{X}$ is formed by 2^K global states.



Type	Event	Rate
loc	et_k	μ_k
syn	te_k	λ_k

Fig. 4. Philosophers model without reservation

$\mathcal{X}^M$ extraction Supposing six philosophers in a table (Figure 5) the application of the transition function returns the extremal states for the SAN model. Regarding structural properties of this model all events $et_k, te_k \in \xi$ are monotone since they retain the component-wise ordering of global states in the chain formed by this class of models. Algorithm 3 finds the last states $\tilde{s} \in \mathcal{X}$ that can be accessed, *i.e.*, the extremal states $\tilde{s} \in \mathcal{X}^M$. Then component-wise property allows the feasible set formation based on indexes, because the successive application of events (tracing a trajectory) leads to a state where it is not possible to go on to a greater state, *i.e.*, they are the extremal states of the chain. For these states there are events just to go back in the paths already generated.

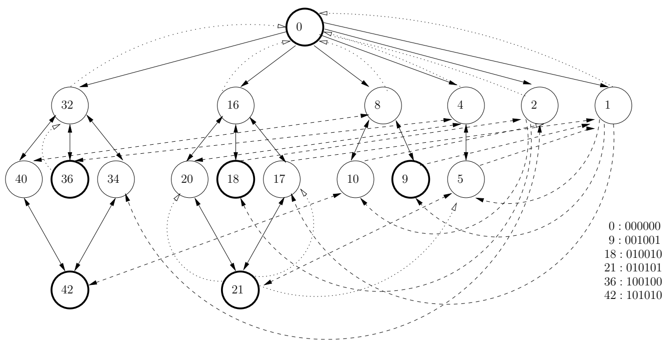


Fig. 5. Ordered $\mathcal{X}^R$ supposing 6 philosophers

Considering the model global states formed by *bits* in the Figure 5, since the state $T^{(k)}$ is represented by O and $E^{(k)}$ represented by 1, we have for example, a set $\mathcal{X}^R = 18$ and $\mathcal{X}^M = 6$ (the marked states are maximal elements) for a model with six philosophers. Table 2 shows the SANmodel with $K = 6 \dots 26$ and their respective $\mathcal{X}$, $\mathcal{X}^R$ and the extracted set of extremal states $\mathcal{X}^M$. Since the size of the model grows exponentially the size of maximal set grows slowly comparatively. The size of $\mathcal{X}^M$ is the number of trajectories to run in parallel, *i.e.*, the number of vector positions to store, so it is also the computational cost in memory positions to run perfect simulation. Each position is an integer representing the current global state index in the trajectory. When we have more philosophers in the model the impact of this optimization is more clear mainly verifying the time spent to solve this models using *PEPS* software tool and the perfect simulation module (*Perfect PEPS*).

K	$\mathcal{X}$	$\mathcal{X}^R$	$\mathcal{X}^M$	*PEPS* (iteration)	*Perfect PEPS* (sample)
6	64	18	6	0.000004 sec.	0.002711 sec.
8	256	47	11	0.000123 sec.	0.003464 sec.
10	1,024	123	18	0.000542 sec.	0.005682 sec.
12	4,096	322	30	0.002487 sec.	0.012290 sec.
14	16,384	843	52	0.011832 sec.	0.029745 sec.
16	65,536	2,207	91	0.055973 sec.	0.074337 sec.
18	262,144	5,778	159	0.296355 sec.	0.184355 sec.
20	1,048,576	15,127	278	1.394022 sec.	0.457599 sec.
25	33,554,432	167,761	1,131	5.115790 sec.	4.736356 sec.
26	67,108,864	392,836	2,779	—-	13.500322 sec.

Table 2. Resource sharing without reservation

The actual number of samples to generate depends immensely on the numeric characteristics of the model itself. Different parameters as the actual numeric rates of the events, may change the required number of samples to achieve statistical convergence of the stationary prediction. Analogously, the numbers of iterations to perform the iterative solution methods in the *PEPS* tool also depends on the model numeric characteristics. Therefore in the Table 2 we indicate the amount of time needed to perform one single sample generation with the contracted state space in the *Perfect PEPS* module, and one single iteration in the numerical solution implemented by *PEPS*. The presented values in seconds must be considered with caution, since nothing relates the number of needed iterations in *PEPS* with the number of samples needed in our simulation tool. For example, the first model ($K = 6$) needed 528 iterations to achieve a precision of 10^{-10} in the *PEPS* solver. We obtain the simulation results running a fixed number of $100,000$ samples. However, a smaller number of samples would probably already be enough to achieve (statistically) the required precision for such small example using confidence intervals. The example was extended just beyond the capacity limit of *PEPS*, since the last example ($K = 26$) is already too huge to run on our target machine that holds problems of as many as 64 million states.

4.2 Dining Philosophers with Reservation

We can extend the mutual exclusion in resource sharing models to analyse more deeply the locking of shared resources in systems. But here the goal is to obtain an extensible model where a numerical solution is no longer possible due state space explosion, in order to show the possible product state space contraction also in these cases.

Figure 6 has K automata $P^{(k)}$ representing the philosophers, each one with three ordered states: $T^{(k)}$ (thinking), $L^{(k)}$ (taking left fork), $R^{(k)}$ (taking right fork). The philosopher can reserve the fork on his immediate left or right waiting for eating with two available forks. To avoid deadlock is established an ordering to get the forks in the table, for each philosopher in the model. The product state space $\mathcal{X}$ is formed by 3^K global states.

Type	Event	Rate
loc	lt_i	μ
syn	tr_i	λ
syn	rl_i	λ
loc	rt_K	μ
syn	tl_K	λ
syn	lr_K	λ

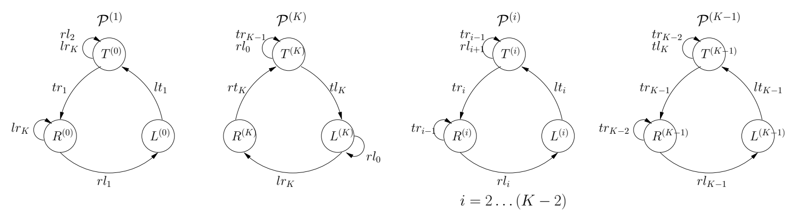


Fig. 6. Philosophers SAN Model with Reservation

$\mathcal{X}^M$ extraction Regarding structural properties of this extended model, the monotonicity properties are also maintained for all new events generated lt_i, tr_i, rl_i, rt_k, tl_k and lr_k. The inclusion of a new state in each automaton and new events constraints does not interfere in $\mathcal{X}$ partial ordering. Since the minimal global state 0 (all philosophers thinking) is the initial state to generate the feasible set of the model, the extremal states are naturally the ones with greater indexes than their consequent transitions.

Table 3 shows in its last lines, huge models to solve with *PEPS* software tool mainly because the size of $\mathcal{X}$, and the possible contraction of state space in $\mathcal{X}^M$ to run perfect simulation. The costs in memory are drastically reduced since for monotone versions we used to store just the coupling vector with extremal elements instead of the product state space. The same remarks still apply to the times presented here, specially the fact that this table present times for one iteration in the *PEPS* numerical solution, and one sample generation for *Perfect PEPS*. For the last model ($K = 18$) the *PEPS* solution could not be achieved since it represents a state space of more that 327 million states, which is considerably above the current overall numerical solution limitation that is a little below 100 million states in a 4GBytes memory machine.

5 Conclusions

We show that is possible to design a perfect sampling algorithm for SAN through backward coupling. For the underlying Markovian graphs, the simulation coupling time can be greatly reduced by using extremal initial states to run trajectories in parallel. In fact, this paper not even present the times for sample generation without using the state space contraction because even the average models, *e.g.*, Resource Sharing without reservation $K = 14$, would represent a model much slower than the larger model we

K	$\mathcal{X}$	$\mathcal{X}^R$	$\mathcal{X}^M$	*PEPS* (iteration)	*Perfect PEPS (sample)*
5	243	70	11	0.000130 sec.	0.004547 sec.
6	729	169	17	0.000474 sec.	0.007829 sec.
7	2187	408	27	0.001693 sec.	0.014273 sec.
8	6,561	985	43	0.003185 sec.	0.032354 sec.
10	59,049	5,741	111	0.038100 sec.	0.111365 sec.
12	531,441	33,461	289	0.551290 sec.	0.689674 sec.
14	4,782,969	195,025	755	5.712210 sec.	2.686925 sec.
16	43,046,721	1,136,689	1,975	68.704325 sec.	15.793501 sec.
18	387,420,489	6,625,109	5,169	—-	83.287321 sec.

Table 3. Resource sharing with reservation

were able to solve (Resource Sharing with reservation $K = 18$). However, the study of coupling times considering the maximal set $\mathcal{X}^M$ is a work in progress. Preliminary results shows that for the models which the numerical solution is no longer possible with *PEPS*, perfect simulation seems to be a reasonable alternative. The natural future work in that direction is a study of the effects of our technique in the statistical convergence. Such study could also compare the time of a full stationary solution using perfect simulation and the traditional numerical solution.

The current limitation of *PEPS* software tool is in order of $\mathcal{X} \leq 6 \times 10^7$ states using 1 Gbyte RAM machine because it needs to store probability vectors for the state space $\mathcal{X}$. SAN simulation approaches, on the contrary, will work only with vectors of size related to maximal sets of the models. Even further, in the case we have a particular interest in a given aggregation function of result, *e.g.*, compute the probability of a local state, our solution may avoid to store the probability vectors for the maximal sets, but only compute the aggregation function over the generated samples. This last improvement combined with the monotonicity property identified in models with a component-wise order can help us to solve really huge models. In fact, this approach may, virtually, have no size bound, since not the transition matrix, nor the probability vector can be stored. Such possible applications will only have a time bound to be considered, and since the generation of samples can be performed in parallel, even this time bound can be overcome for models with thousands of millions states. Only the statistical analysis of the generated samples will have to be dealt. Nevertheless, all these reasons let us believe that the perfect simulation of SAN with component-wise ordering allowing state space contraction is a more than worthy solution when the use of numerical methods is just not possible with the current technology.

Acknowledgment

This work is partially supported by the brazilian government (CAPES grants 1341/07-3 and 2265/06-0), FINEP project STGSD grant nb. 4284/05 and ANR SETIN Checkbound and ANR blanche SMS. Author Thais Webber is invited by the Project MESCAL-INRIA (Thais.Webber@imag.fr).

References

1. Stewart, W.J.: Introduction to the numerical solution of Markov chains. Princeton University Press (1994)
2. Plateau, B.: On the stochastic structure of parallelism and synchronization models for distributed algorithms. In: ACM Sigmetrics Conference on Measurements and Modeling of Computer Systems, USA, ACM Press (1985) 147–154
3. Benoit, A., Fernandes, P., Plateau, B., Stewart, W.J.: On the benefits of using functional transitions and Kronecker algebra (In Press). Performance Evaluation (2004)
4. Buchholz, P., Ciardo, G., Donatelli, S., Kemper, P.: Complexity of memory efficient Kronecker operations with applications to the solution of Markov models. INFORMS Journal on Computing **13**(3) (2000) 203–222
5. Fernandes, P., Plateau, B., Stewart, W.J.: Efficient vector-descriptor multiplication in Stochastic Automata Networks. Journal of the ACM **45**(3) (1998) 381–414
6. Buchholz, P., Kemper, P.: Hierarchical reachability graph generation for Petri nets. Formal Methods in Systems Design **21**(3) (2002) 281–315
7. Miner, A.S., Ciardo, G.: Efficient Reachability Set Generation and Storage Using Decision Diagrams. In: Proceedings of the 20^{th} International Conference on Applications and Theory of Petri Nets. Volume 1639 of LNCS., USA (June 1999) 6–25
8. Jungblut-Hessel, R., Plateau, B., Stewart, W.J., Ycart, B.: Fast simulation for Road Traffic Network. RAIRO Operational Research **35** (2001) 229–250
9. Häggström, O.: Finite Markov Chains and Algorithmic Applications. Cambridge University Press (2002) Based on lecture notes.
10. Propp, J.G., Wilson, D.B.: Exact Sampling with Coupled Markov Chains and Applications to Statistical Mechanics. Random Structures and Algorithms **9**(1-2) (1996) 223–252
11. Plateau, B., Atif, K.: Stochastic Automata Networks for modelling parallel systems. IEEE Transactions on Software Engineering **17**(10) (1991) 1093–1108
12. Brenner, L., Fernandes, P., Sales, A.: The Need for and the Advantages of Generalized Tensor Algebra for Kronecker Structured Representations. In: 20^{th} Annual UK Performance Engineering Workshop, Bradford, UK (July 2004) 48–60
13. Glasserman, P., Yao, D.D.: Monotone structure in discrete-event systems. John Wiley & Sons, Inc., New York, NY, USA (1994)
14. Borovkov, A.A., Foss, S.G.: Two ergodicity criteria for stochastically recursive sequences. ACTA Applicande Math **34** (1994) 125–134
15. Stenflo, O.: Ergodic Theorems for Markov chains represented by Iterated Function Systems. Bull. Polish Acad. Sci. Math. **49**(1) (2001) 27–43
16. Benoit, A., Brenner, L., Fernandes, P., Plateau, B., Stewart, W.J.: The PEPS Software Tool. In: Computer Performance Evaluation. Volume 2794 of LNCS., USA (2003) 98–115
17. L.Brenner, P.Fernandes, B.Plateau, I.Sbeity.: PEPS2007 - Stochastic Automata Networks Software Tool. In: QEST 2007, IEEE Press (2007) 163–164
18. Vincent, J.M.: Perfect simulation of monotone systems for rare event probability estimation. In: Winter Simulation Conference, ACM (2005) 528–537
19. Vincent, J.M.: Perfect Simulation of Queueing Networks with Blocking and Rejection. In: International Symposium on Applications and the Internet Workshops: SAINT Workshops, IEEE Computer Society (2005) 268–271
20. Vincent, J.M., Vienne, J.: Perfect simulation of index based routing queueing networks. SIGMETRICS Performance Evaluation Review **34**(2) (2006) 24–25
21. Dimakos, X.K.: A Guide to Exact Simulation. International Statistical Review **69**(1) (2001) 27–48