Stationary Solution Approximation using a Memory-Efficient Perfect Sampling Technique

Ricardo M. Czekster, Paulo Fernandes, Afonso Sales, and Thais Webber
Pontifícia Universidade Católica do Rio Grande do Sul
Avenida Ipiranga, 6681 – 90619-900 – Porto Alegre – RS – Brazil
{ricardo.czekster, paulo.fernandes, afonso.sales, thais.webber}@pucrs.br

Keywords: Discrete event simulation, Perfect sampling, Markovian models, SAN, Integration functions

Abstract
The analytical solution of large Markovian models is one of the major challenges in performance evaluation. Structured formalisms provide a modular description to tackle state space explosion by presenting memory-efficient solutions based on tensor algebra and specific software tools implement such solutions using iterative methods. However, even these numerical methods become unsuitable when massively large models are considered, i.e., models with more than 100 million states. To deal with such classes of models is possible to find approximations of the stationary solution using simulation of long-run trajectories with perfect sampling methods. The use of these methods prevents usual simulation problems such as initial state setup and burn-in time. Unfortunately, the number of produced samples to establish statistically significant solution remains an open problem. This paper analyzes the sampling process in its extent, proposing a memory-efficient stopping criteria based on a numerical tolerance of the measures of interest as well as on tensor algebra. However, the current numerical iterative solutions, our proposed simulation approach also copes with memory bottlenecks related to large state spaces, while it keeps nearly the same precision of measures of interest. Additionally, the processing time has no relevant overhead because the measures computation is usually quite light when compared to the cost of sample generation itself.

The main contribution of this paper is the proposal of a memory-efficient stopping criteria to the sample generation based on a numerical tolerance of the measures of interest as the number of samples increases. In the context of Markovian modeling, the measures of interest are also known as integration functions, which are summations of specific elementary probabilities from the steady-state vector. Related works already used integration functions as optimization, specifically for coupling reduction purposes [13]. In comparison with numerical iterative solutions, our proposed simulation approach also draws considerations about the method itself and future work about the parallelization of the proposed method.

1. INTRODUCTION
The difficulty in constructing and characterizing the states of systems such as parallel and distributed computer systems or other applications non specific to computer science is followed by the complexity to analytically solve these models. The first step involved in calculating measures such as performance indices of discrete state space systems is to build a Markov chain [1] characterizing the states and describing the manner in which they move, from one state to another. Stochastic Automata Networks (SAN) formalism [2, 3] provides a modular description diminishing the state space explosion by allowing memory-efficient solutions based on tensor algebra. However, the current numerical iterative solution implemented in specific software tool (such as PEPS [4] and GTAEXPRESS [5]) is insufficient to cope with high memory requirements and processing constraints needed for models with more than 100 million states.

Simulation is a widely used and a popular method for studying complex systems [6]. Discrete-event simulation approaches [7, 8, 9] are often used to estimate an approximation of the steady-state behavior of systems, providing samples for later statistical analysis. However, the drawbacks of these techniques [6], such as the initial state setup and the burn-in time, can produce biased samples. New algorithmic solutions based on backward coupling, namely Coupling From The Past (CFTP), overcome these problems generating confident samples from the stationary distribution [10]. This principle was adapted to SAN [11] as well as to other Markovian structured formalisms [7, 12]. Unfortunately, the number of necessary samples to obtain a good approximation of a measure of interest is still a source of concern.

This paper is organized as follows: Section 2 describes the system modeling issues, input parameters and measures of interest. Section 3 introduces the principles of perfect sampling and explains the advantages of models with a component-wise property for memory cost reduction. Section 4 presents the proposed method to generate samples in backward simulation, respecting a numerical tolerance of the measures of interest. In this section, we present a model for the resource allocation analysis in two configuration in order to discuss precision and memory cost results. Finally, the conclusion draws considerations about the method itself and future work about the parallelization of the proposed method.
2. SYSTEM MODELING

Stochastic Automata Networks (SAN) proposes a description framework with a modular way to model and compute performance indices of continuous and discrete-time Markovian models [2, 3]. The SAN basic idea is to represent a whole system by a collection of K subsystems described as a set of stochastic automata \( \{ \mathcal{A}^{(1)}, \ldots, \mathcal{A}^{(K)} \} \). An automaton \( \mathcal{A}^{(k)} \) with \( k \in [1..K] \) has a set \( \delta^{(k)} \) of local states \( s_k^{(k)} \), where \( i \in [1..n_k] \), \( n_k = |\delta^{(k)}| \), and \( |\delta^{(k)}| \) is the cardinality of set \( \delta^{(k)} \), i.e., \( n_k \) is the total number of states of automaton \( \mathcal{A}^{(k)} \).

The network of automata presents a constant set of \( P \) events called \( \xi = \{ e_1, \ldots, e_P \} \), where each event includes probabilistic and timing information to the state transitions. In an automaton the states are interconnected by transitions labeled by events \( e \in \xi \).

An event can appear in more than one automaton meaning a synchronization among subsystems. Such events are called synchronizing events. In opposition, events that appear in only one automaton represent a change in one single subsystem. This events are called local events.

Besides the synchronizing event, an additional form of interference among subsystems is the definition of functional rates. In continuous-time models, each event \( e \) must have a rate \( \lambda \) that is the average of the exponentially distributed frequency of occurrence of the event. This rate can be either a constant value meaning that this event always behave the same during the system evolution, or a function that depends on the local states of automata of the model [14, 3].

A global state in a SAN model is a snapshot of local states in every component, i.e., a global state \( \tilde{s} \) of a SAN with \( K \) automata is a vector \( \tilde{s} = \{ s^{(1)}, \ldots, s^{(K)} \} \) where each automaton \( \mathcal{A}^{(k)} \) is at the local state \( s^{(k)} \in \delta^{(k)} \). Set \( X \) of a SAN model is defined by the Cartesian product of all set of local states of all automata, also called product state space. There is a subset of \( X \), called reachable state space \( X_{R}^{\tilde{s}_0} \), which indicates the consistent (valid) state space related to a model, i.e., all states inside this set are reachable from a given initial state \( \tilde{s}_0 \). This set is an irreducible component obtained from state \( \tilde{s}_0 \in X \) by successive firing of events in \( \xi \), and it can be efficiently generated and stored using multi-valued decision diagrams [15].

2.1. Integration functions

The objective of analytical models is to generate one or more measures derived from the steady-state vector \( \pi \), which contains the probability results of the model. The vector \( \pi \) of size equal to \( |X| \) has \( |X| \) nonzero values related to the reachable states probabilities achieved mathematically analyzing the model.

A SAN model can specify \( F \) measures of interest as integration functions \( \mathcal{F}^{(i)} : X \rightarrow \mathbb{R} \), where \( i \in [1..F] \), i.e., each of these functions associates a global state space to a Real value.

For traditional iterative solutions the first step is to compute the probability vector \( \pi \) which is the stationary solution of the model. Once this is done, the integration function is computed as the sum, for all global states in \( X \), of the product of the integration function result and the probability of the corresponding global state in \( \pi \). Formally, the \( i \)-th measure of interest, denoted by \( \gamma^{(i)} \), is computed as:

\[
\gamma^{(i)} = \sum_{\forall \tilde{s} \in X} \mathcal{F}^{(i)}(\tilde{s}) \times \pi_{\tilde{s}}
\]

The computation of the measures of interest is not very complicated, but it is necessary to store the probability vector \( \pi \) and, more than that, traditional iterative methods [1] use at least another vector of equal size (\( \mathcal{X} \)) to compute the vector \( \pi \). This is certainly a bottleneck for huge models.

If the simulation solution, on the contrary, there is no need to directly compute probability vectors. Only samples of global states within the stationary solution are generated at each step. A naive, and yet very common, approach to compute measures of interest is to annotate the frequency of generated samples until satisfaction and then normalize the results to obtain the probability vector \( \pi \). Hence, the integration functions can be computed as for iterative solutions.

Obviously this simulation has several advantages in comparison with the iterative numerical approach, since only one vector needs to be stored. However, for huge models, this vector can be prohibitive in a storage point of view. Integration functions are suitable to obtain numerical solutions by simulation, since they are extremely compact to store in memory.

2.2. System evolution rules

The complexity and drawbacks to describe a SAN as a core for a discrete-event simulation based on the system rules is earlier discussed in [9]. The rules in which the system evolves, i.e., the transition function is more complex in structured models because many automata can change their local states independently or in a synchronizing manner.

The proposal is to observe the subcomponents of the system in a global fashion without a hierarchy to cope with in the simulation core, i.e., the hierarchy is only implicitly maintained. This means the simulation algorithm must control the events activation in all involved automata at the same time, indicating that an event \( e \in \xi \) is enabled or disabled for each local state \( s^{(k)} \in \delta^{(k)} \). Moreover, if \( e \in \xi \) is a synchronizing event, its activation must consider the global perspective. If the event is considered disabled, the system executes a self-transition in the current global state \( \tilde{s} = \{ s^{(1)}, \ldots, s^{(K)} \} \). A global state update can be viewed generically as \( \phi(\tilde{s}, e) = \tilde{r} \) if \( e \) is enabled for \( \tilde{s} \), or \( \phi(\tilde{s}, e) = \tilde{s} \) if \( e \) is disabled for \( \tilde{s} \).

The system evolution is described by a stochastic recursive sequence [16, 17, 10], a general transition function \( \phi \) which is typically given by:

\[
\tilde{s}_{n+1} = \phi(\tilde{s}_n, e_{n+1})
\]
where $\tilde{s}_n$ is the $n^{th}$ observed state of the system and $\{e_n\}_{n \in \mathbb{N}}$ the sequence of generated events. Let $\xi_{\tilde{s}_n}$ be the set of enabled events for $n^{th}$ observed state $\tilde{s}_n$ and let $\rho$ be the cardinality of this set, i.e., $\rho = |\xi_{\tilde{s}_n}|$. The application of each event $e$ is according the distribution $\left(\frac{\lambda_1}{\Lambda}, \ldots, \frac{\lambda_n}{\Lambda}\right)$, where $\Lambda = \sum_{i=1}^{\rho} \lambda_i$ is used to uniformize the rates of events $e \in \xi_{\tilde{s}_n}$. If the sequence of generated events is independent and identically distributed, the process $\{\tilde{s}_n\}$ defined by an initial value $\tilde{s}_0$ and the recursive equation (2) is a Markov chain.

The system evolution can be followed through forward steps, tracing a trajectory [6]. In traditional discrete event simulations, the events are successively applied to a current global state $\tilde{s}$ due to obtain samples of the stationary distribution. These techniques have some problems related to the definition of the initial state and also the number of iterations needed to reach the stationary regime, called burn-in time [6]. Next section we explain a simulation method to overcome these drawbacks related to the traditional simulation technique - Figure 1 (a).

3. PERFECT SAMPLING IN SAN

Propp and Wilson [10] proposed a method named backward coupling simulation or CFTP (Coupling From The Past), where the problem of biased simulations in traditional perfect sampling methods is completely solved. Also called as perfect sampling the simulation process enables us to compute samples exactly distributed according to the steady-state distribution of the Markov process. Running trajectories in parallel, starting from all possible states, the coupling of trajectories in a time $\tau$ generates what is called an exact sample $\tilde{s}_\tau$ [6].

Given the set of reachable states $X^R_{\tilde{s}_0}$ of a SAN, a set $\mathcal{E}_{\tilde{s}_0}$ of generated events and the transition function $\phi : X^R_{\tilde{s}_0} \times \mathcal{E}_{\tilde{s}_0} \to X^R_{\tilde{s}_0}$, issuing from all global states of $X^R_{\tilde{s}_0}$ we know that they will couple in a state $\tilde{s}_\tau$ for a given sequence of events $\mathcal{E}_{\tilde{s}_0}$.

The problem related to the definition of an initial state $\tilde{s}_0$ is solved when it is possible to start a simulation issuing from all reachable global states of a model. The problem of determining burn-in time is avoided running trajectories backwards in time allowing the collection of exact samples. In Figure 1 (b), all trajectories issued from all reachable states at time $\tau_0$, coupled in a given state at time $\tau$. Since the coupling time $\tau$ is almost surely finite, the backward scheme provides a sample distributed according to the steady-state distribution because the method determines automatically when to stop [18, 10].

In Algorithm 1, we present a Perfect Sampling algorithm applied to SAN, where the vector $\omega$ is initialized with all global states $\tilde{s} \in X^R_{\tilde{s}_0}$ at simulation time $\tau_0$ (line 2). At each simulation iteration, one event is generated (line 6) and the related transition functions are applied to each position of $\omega$ (line 9). Each new state generated actually indices the vector $\omega$, which has the last version of $\omega$ stored. This process is called backward coupling because we compute $\omega(\tilde{s})$ at time $\tau_0$ of trajectory issued from $\tilde{s}$ at time $\tau$. This procedure will be repeated until all positions of vector $\omega$ have the same result, i.e., the same global state in all positions, where all trajectories running in parallel have coupled (line 11).

Algorithm 1 PerfSampling(): Perfect Sampling in SAN

1: for all $\tilde{s} \in X^R_{\tilde{s}_0}$ do
2: $\omega(\tilde{s}) \leftarrow \tilde{s}$ {initializing trajectories with global states}
3: end for
4: repeat
5: $\omega_0 \leftarrow \omega$ {saving the state of each trajectory}
6: $e \leftarrow \text{Generate-event}( \phi(\tilde{s}, e))$ {random generation of $e$ according the distribution $\left(\frac{\lambda_1}{\Lambda}, \ldots, \frac{\lambda_n}{\Lambda}\right)$}
7: {computing $\omega(\tilde{s})$ at time $\tau$ of trajectory issued from $\tilde{s}$ at a time $\tau$ in the past}
8: for all $\tilde{s} \in X^R_{\tilde{s}_0}$ do
9: $\omega(\tilde{s}) \leftarrow \omega_0(\phi(\tilde{s}, e))$
10: end for
11: until global states are equal in all trajectories {current time $t$ is the backward coupling time $\tau$}
12: return $\omega(\tilde{s})$ {generated sample}

3.1. Monotonicity in SAN

The memory needed for running perfect sampling considers the storage of vectors of $|X^R_{\tilde{s}_0}|$ long int values, i.e., one for coupling trajectories and another for collecting samples.
statistics. The needed size in memory is given by \(2 \times |X_R^{0}|\). Simulation methods can bring a memory optimization to solve models when compared to traditional iterative numerical solutions. Iterative solvers spend at least \(3 \times |X|\) (two vectors for the iterative method and one to store the stationary probabilities), even for models when \(|X_R^0| < |X|\) in the usual tools for SAN solution. However, the size of \(X_R^0\) can be exponential depending on the size of the SAN model and it can be really huge to generate and to deal with. Soon, memory and processing time become also a bottleneck for perfect sampling methods.

Related studies of monotonicity in Markovian models [7, 12] resulted in monotonic backward coupling algorithms that profit of state spaces partial orderings to reduce the coupling vector and time to coupling. The monotonicity of a SAN model can be verified looking at the set \(\mathcal{X}\) where an event \(e \in \mathcal{X}\) is said to be monotone if it preserves the partial ordering \((<)\) on \(X\). That is \(\forall (s_1, s_2) \in \mathcal{X}, s_1 < s_2 \Rightarrow \phi(s_1, e) < \phi(s_2, e)\). If all events in \(\mathcal{X}\) are monotone, the global system is said to be monotone too.

The monotonicity property [18] states that considering an identified partial order of \(\mathcal{X}\) and an extremal set \(X^M \subseteq X_R^0\), if all trajectories issued from the extremal set \(X^M\) coupled at time 0, then they will also couple with all other states in \(X_R^0\). Note that the canonical component-wise ordering \((|X^M| = 2)\) and the non-lattice component-wise ordering \((|X^M| < |X_R^0|)\) are kinds of SAN partial ordering applied to run monotone backward simulation versions [11]. However it is not always easy to identify these extremal global states inside \(X_R^0\), but once they are identified the coupling vector can be drastically reduced and, consequently, the coupling time and the memory needed to run parallel trajectories are minimized.

The proposed constructive algorithm (Algorithm 2) analyzes each reachable state of the SAN model (lines 4-20), firing events of set \(\mathcal{X}\) (lines 6-13) and storing the generated states at each firing (line 10). The search for extremal elements starts from an initial state \(s_{\text{min}}\) (line 1) defined by a component-wise ordering already known (which can be simply a lexicographical order). This algorithm resembles the classical procedure for topological sort [19] to indicate precedences among events in a directed acyclic graph.

Observe that event firings from a state can lead to the same state (or reachable states already accessed). In this case, the observed state can be an extremal state (lines 15-17), i.e., this classification happens if the state does not generate any new state from the firing of all possible events in the model. Note that, by the same principle, transitions fired from the established minimal state do not achieve states lower than itself, considering we already started from the canonical minimum. The cardinality of the extremal set \(X^M\) is not determined until the end of this constructive process and it is an open research the reduction of this set.

### Algorithm 2: Extremal Set for a Class of SAN Models

1. \(M \leftarrow \text{Add}(s_{\text{min}})\) \{(list of accessed states storing an initial state \(s_{\text{min}}\)\)
2. \(cState \leftarrow M[1]\) \{(it indicates the current observed state in the list \(M\)\)
3. \(i \leftarrow 1\) \{(it counts the total number of accessed states\)
4. repeat
5. isExtremal \(\leftarrow\) true
6. for all \(e \in \mathcal{X}\) do
7. \{(events firing over the current observed state\)
8. nState \(\leftarrow \phi(cState, e)\)
9. if \((\text{nState} \notin \{M[1], \ldots, M[cState]\})\) then
10. \(M \leftarrow \text{Add}(\text{nState})\) \{(if it is not already accessed, it adds new state to the list \(M\) and it is not an extremal\)
11. isExtremal \(\leftarrow\) false
12. end if
13. end for
14. \{(if no nState is added to the list \(M\), cState is an extremal\)
15. if (isExtremal) then
16. \(X^M \leftarrow \text{Add}(\text{cState})\) \{(it adds cState to \(X^M\)\)
17. end if
18. \(i \leftarrow i + 1\) \{(the next element in \(M\) to analyze firings\)
19. cState \(\leftarrow M[i]\) \{(it updates current state\)
20. until \(i = |X_R^0|\) \{(access all reachable states\)
21. return extremal set \(X^M\) \{(the extremal set is completed\)

The monotone version of Algorithm 1 uses a coupling vector \(\omega\) of \(|X^M|\) positions running these trajectories in parallel [11]. Also, it needs to store the events generated in the whole trajectory, because it uses a doubling scheme [10] to generate and apply these events in each trajectory. A doubling scheme is a simple procedure that verifies for a given number of events if the coupling does not occur, then doubling this number of events for a new simulation run. The memory spent for running the monotone algorithm is highly related to the cardinality of \(X^M\) (which states the size of the coupling vector) and the size of \(X_R^0\) (the vector to collect samples statistics), i.e., \(|X^M| + |X_R^0|\) needed positions in memory.

4. MEMORY-EFFICIENT STOPPING CRITERIA

Statistical techniques can be used to analyze the output data of simulation runs. Structured analytical models have sometimes thousands of reachable states collected during these experiments, and often the output becomes a bottleneck for memory resources.

Collected samples compose the probability vector \(\pi\) related to the steady-state (or an approximation of the stationary distribution) used to calculate the model measures or performance indices. The probability vector \(\pi\) has in fact \(|X_R^0|\) nonzero values related to the reachable configurations achieved mathematically solving the model. Usually, an analytical SAN model can also specify integration functions to
convert the state space sized probability vector \( \pi \) on estimated measures about the system.

The memory for storing the probability vector \( \pi \) is actually an extra bound for iterative solution methods since measures based on the stationary distribution are calculated only after the convergence of the method. For simulation, it also becomes a bottleneck, mainly using non-monotone versions where the memory needed for the coupling vector is potentially related to the size of \( X_0^\infty \).

To overcome these memory constraints, we present, in Algorithm 3, a memory-efficient stopping criteria to the perfect sample generation based on a numerical tolerance for the integration functions \( \mathcal{F} \) of the model. The main idea of this algorithm is to evaluate an integration function for a large number of samples (\( \eta \) times) and, after this warm up period of the function, adopt as stopping criteria a given numerical tolerance \( \varepsilon \) between two evaluations of this function, considering the generation of batches of \( \kappa \) samples, where \( \kappa \ll \eta \).

**Algorithm 3** Perfect sampling using a memory-efficient stopping criteria

1. \( \gamma \leftarrow 0 \)
2. for \( i \leftarrow 1 \) to \( \eta \) do
3. \( \hat{\gamma} \leftarrow \text{PerfSampling}() \) \( (\hat{\gamma} \) generated sample obtained by Algorithm 1\)
4. \( \gamma \leftarrow \gamma + \mathcal{F}(\hat{\gamma}) \) \( (\gamma \) function evaluation of sample \( \hat{\gamma} \) accumulated in \( \gamma \)\)
5. end for
6. \( b \leftarrow 0 \)
7. repeat
8. \( \gamma_o \leftarrow \gamma \)
9. for \( i \leftarrow 1 \) to \( \kappa \) do
10. \( \hat{\gamma} \leftarrow \text{PerfSampling}() \)
11. \( \gamma \leftarrow \gamma + \mathcal{F}(\hat{\gamma}) \)
12. end for
13. \( b \leftarrow b + 1 \)
14. until \( \frac{\gamma}{\eta + (b \times \kappa)} \leq \varepsilon \)
15. \( (\varepsilon \) is the tolerance accepted between two observed measures of \( \mathcal{F} \)\)
16. return \( \frac{\gamma}{\eta + (b \times \kappa)} \) \( (\gamma \) measure observed about a model\)

In Algorithm 3 (lines 3 and 10), the samples are generated using Algorithm 1 (in a monotone or non-monotone version). The first step in the algorithm (lines 2-5) is the generation of \( \eta \) samples, evaluating them for the integration function \( \mathcal{F} \) and accumulating the results on \( \gamma \). After the warm up of the function, \( \gamma_o \) saves the integration function evaluation results (line 8) and batches of \( \kappa \) samples are performed (lines 9-12). The function evaluation process executed for each \( \kappa \) samples is repeated until a given numerical tolerance \( \varepsilon \) is achieved (line 14), i.e., the difference between the two observed measures of \( \mathcal{F} \) (\( \gamma \) and \( \gamma_o \)) is lower than \( \varepsilon \). Once the tolerance is accepted, the measure of interest \( \mathcal{F} \) for all generated samples, i.e., \( \eta + (b \times \kappa) \) samples, is computed (line 16).

Despite the simplicity of Algorithm 3, the use of integration function instead of a (large) vector allows the perfect sampling of huge models, where the solution is no longer possible using traditional iterative solvers and the memory cost for backward coupling becomes high enough to be suitable.

In the next section we discuss in detail numerical implications and memory costs issues to implement perfect sampling and collect measures of interest in a memory-efficient way.

### 4.1. Numerical analysis and memory costs

We present in this section two models described in the Stochastic Automata Networks (SAN) formalism in order to discuss some issues about numerical precision and memory costs for the models simulation.

The first model analyzes the resource allocation problem of \( K \) processes that are running applications or requiring external resources. This model resembles the classical dining philosophers model (Figure 2), where each philosopher (process) is represented by an automaton \( P \) with two states: thinking (state \( T \)) and eating (state \( E \)). Supposing that they sit at a circular table with a large bowl of food in the center, a fork (or resource) is placed between each philosopher. So, each philosopher has one fork to his/her left and one to his/her right side. The philosopher must have two forks (at the same time) to eat. Forks in this scenario represent the external resources needed to complete an operation. Only one process can access each resource in a given time, so this configure a mutual exclusion regime.

**Figure 2.** Allocation model without resource reservation.

In Figure 2, state \( E \) supposes the philosopher required both forks at the same time to eat, without reserving earlier the first fork to then take the second. The model presents \( K \) synchronizing events with constant rates for eating (in \( K \) local events \( e_{T_1}..e_{T_K} \)) and \( K \) local events \( (e_{T_1}..e_{T_K}) \) representing the release of forks returning to thinking state. The product state space \( X \) of this model is composed of \( 2^K \) states.

The model presents two measures of interest to be calculated, for example \( (F = 2): \mathcal{F}^{(1)} \) is the integration function related to having all philosophers (automata) in state \( T \), and \( \mathcal{F}^{(2)} \) is the integration function to calculate the probability...
when only the first philosopher (automaton) is in state $E$ (i.e., only the first philosopher is eating).

The measures observed for the model are more accurate when the number of acquired samples increases. This means that at each batch of generated samples (for instance, $\kappa \geq 10,000$), the indices are more reliable. When using simulation methods, a confidence interval with 95% (related to the execution of simulation batches of samples) indicates a reliable approximation of the model steady-state solution.

Considering $K = 6$ philosophers to the model depicted in Figure 2, we compute the integration functions $\mathcal{F}^{(1)}$ and $\mathcal{F}^{(2)}$ using a specific software tool (PEPS [4]) through an iterative method (e.g., Power method), and the results were 5.56% and 27.77% respectively (i.e., the probability of having all philosophers in state $T$ was 5.56%, and the probability of only the first philosopher be in state $E$ was 27.77%). In order to validate our integration function results (by the usage of Algorithm 3), we have used the Chi-square test for 30 experiments. The test is performed in different quantities of samples, and we also used a tolerance $\varepsilon = 10^{-5}$ as simulation stopping criteria. Table 1 presents the results for both methods (backward and monotone backward), which are really close to the iterative method.

<table>
<thead>
<tr>
<th>Method</th>
<th>Integration Functions</th>
<th>$\mathcal{F}^{(1)}$</th>
<th>$\mathcal{F}^{(2)}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iterative</td>
<td></td>
<td>5.56%</td>
<td>27.77%</td>
</tr>
<tr>
<td>Backward</td>
<td></td>
<td>5.58% ± 0.4%</td>
<td>27.71% ± 2%</td>
</tr>
<tr>
<td>Monotone backward</td>
<td></td>
<td>5.55% ± 0.4%</td>
<td>27.80% ± 2%</td>
</tr>
</tbody>
</table>

The average number of samples in the experiments does not follow a given distribution due to the randomness nature of the simulation experiments. But, having a large amount of batches of samples, the result is surely close to the stationary value, since it is recommended to stop the simulation with a high precision achieved between two probability vectors.

Remark that the actual number of samples needed to generate depends immensely on the numeric characteristics of the model itself. Different parameters (e.g., event rates) may change the required number of samples to achieve statistical approximation of the stationary regime. Analogously, the number of iterations to perform the iterative solution also depends on such characteristics. One drawback of the simulation technique is the CPU cost demanded to collect the necessary amount of samples in order to obtain accurate measures about the models. Indeed, there are case studies where an appropriate statistical technique is available [20], but the cost of collecting the quantity of samples dictated by the procedure is prohibitive [6] in sequential executions.

In the case of SAN models simulation, the total time to generate a sample is highly dependable of the transition function definition, which has a great impact on the coupling times. Obviously the coupling time is very reduced when using less global states as initial states. This means we can take advantage of SAN models with a component-wise formulation to obtain better coupling times and also less memory demands. We certainly have an alternative solution using monotonicity properties [11] and integration functions, even if this solution is not time-efficient to solve huge SAN models with a smaller reachable state space, or huge component-wise models with very few extremal states to deal with.

Using a memory-efficient stopping criteria, the processing time spent is only notably increased if we have a huge number of integration functions to obtain the approximated stationary solution. Different integration functions have different quantities of samples needed to achieve the numerical tolerance allowed. Once a given integration function respects this condition, it is no more computed. Then, the generation of more batches of samples gradually reduces the number of integration functions to be evaluated.

Another issue to be analyzed is related to memory costs when using perfect sampling procedures combined with our stopping criteria. For example, we can remodel the dining philosophers problem (Figure 2) to cope with resource reservation rules, where the inclusion of states representing each fork is mandatory. Figure 3 depicts the extension of the SAN model in Figure 2, where $K$ automata represent the philosophers, each one with three states: $T$ (thinking), $L$ (taking left fork), $R$ (taking right fork). The philosopher can reserve the fork on his/her immediate left or right, waiting for eating only if he/she has the two forks. To avoid deadlock is established an ordering to get the forks in the table, for each philosopher in the model, for example, the last philosopher is left-handed.

![Figure 3. Allocation model with resource reservation.](image)

This second model presents a product state space $\mathcal{X}$ of $3^K$ states. However, the reachable state space $\mathcal{X}_R$ is dependent of the number of philosophers ($K$). This model also presents two measures to calculate ($F = 2$) as seen in the earlier ex-
Table 2. Allocation model (Figure 3) with $K = 6$

<table>
<thead>
<tr>
<th>Method</th>
<th>Integration Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$g^{(1)}$</td>
</tr>
<tr>
<td>Iterative</td>
<td>0.06%</td>
</tr>
<tr>
<td>Backward</td>
<td>0.06% ± 0.005%</td>
</tr>
<tr>
<td>Monotone backward</td>
<td>0.06% ± 0.005%</td>
</tr>
</tbody>
</table>

We present in Table 3 the memory costs for the iterative method, backward, and monotone backward simulation method related to the representation of the allocation model with resource reservation (Figure 3). Backward and monotone backward simulation methods use one coupling vector and one probability vector to compute the measures of interest of a model. In this table, besides the memory costs for these (standard) simulation methods, we also present the memory costs for optimized versions of these methods. In these optimized versions, we used only the coupling vector to generate the samples, and the measures of interest of the model are directly computed in a double value in memory.

Table 3. Allocation model memory costs (Figure 3)

<table>
<thead>
<tr>
<th>$K$</th>
<th>$X$</th>
<th>$X_{cT}$</th>
<th>$s^M$</th>
<th>PEPS</th>
<th>Standard model</th>
<th>Optimized model</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Iterative (KB)</td>
<td>Backward (KB)</td>
<td>Monot. (KB)</td>
</tr>
<tr>
<td>6</td>
<td>729</td>
<td>169</td>
<td>17</td>
<td>17.09</td>
<td>1.06</td>
<td>1.30</td>
</tr>
<tr>
<td>7</td>
<td>2,977</td>
<td>408</td>
<td>27</td>
<td>51.26</td>
<td>4.78</td>
<td>3.28</td>
</tr>
<tr>
<td>8</td>
<td>6,561</td>
<td>985</td>
<td>43</td>
<td>153.77</td>
<td>11.54</td>
<td>7.86</td>
</tr>
<tr>
<td>10</td>
<td>59,049</td>
<td>5,741</td>
<td>111</td>
<td>1,385.96</td>
<td>67.28</td>
<td>45.29</td>
</tr>
<tr>
<td>12</td>
<td>531,441</td>
<td>37,461</td>
<td>209</td>
<td>12,495.65</td>
<td>392.32</td>
<td>262.54</td>
</tr>
<tr>
<td>14</td>
<td>4,782,969</td>
<td>101,023</td>
<td>755</td>
<td>112,106.84</td>
<td>2,285.45</td>
<td>1,526.58</td>
</tr>
<tr>
<td>16</td>
<td>43,064,721</td>
<td>1,136,689</td>
<td>1,975</td>
<td>1,008,907.52</td>
<td>13,230.57</td>
<td>8,888.16</td>
</tr>
<tr>
<td>18</td>
<td>387,420,489</td>
<td>6,625,109</td>
<td>5,169</td>
<td>9,080,167.71</td>
<td>77,638.00</td>
<td>51,778.86</td>
</tr>
</tbody>
</table>

Regarding the last configuration ($K = 18$) in Table 3, we notice that the iterative method could not be performed for this configuration, since the model has a state space superior of 387 million states, which is considerably above the current overall numerical solution limitation for a 4 GB memory machine. But, the memory costs are drastically reduced for the optimized versions of backward and monotone backward simulation methods.

A function evaluation spent one double value stored in memory, so the cost to store the (optimized version) solution is only related to the cost of storing $F$ values, where $F$ is the number of integration functions (or measures of interest) of the model. The cost in KB to store the function evaluations is very low even with a large number of integration functions, i.e., with 10,000 functions the memory cost is far from being a problem, considering it is given by the formula $F_c = (\text{size of double} \times F) / 1024$ (if memory expressed in KB).

In a simulation tool, taking as example the allocation model with reservation ($K = 18$), the samples generation could be performed with approximately 20.19 KB at minimum in a monotone version (or around 25,879.33 KB at maximum, using the backward coupling method) to store the coupling vector, plus a factor $F_c$ which indicates the cost to store $F$ integration functions accumulated evaluations. Considering the resource allocation example that has only two integration functions, it needs a very small quantity of bytes to store the simulation results (i.e., to store the computation of the measures of interest).

5. CONCLUSION

The results presented in this paper are an initial effort in order to make feasible the analysis of huge models. Even though much more research remains to be done, the advantages of the proposed approach can already be noticed. The sampling time optimization and a known precision of the simulation experiment are examples of factors achieved. Additionally, the resource allocation models used as example are an abstraction that can be applied generically to many areas such as distributed systems and network protocols performance evaluation with different parameterizations.

Nevertheless, the number of produced samples to observe a measure about a system is still a challenge in the performance evaluation area. Note that the design of a perfect sampling technique applied to structured models presents some solution constraints related to the coupling and solution vectors storage. Monotone versions reduce the size of the coupling vector which is an advantage for models containing this property. SAN models with a component-wise formation also take advantage of routines for extremal states extraction.

The focus of this paper is a memory-efficient simulation algorithm based on integration functions to collect a reasonable amount of samples. Huge models can profit of this approach to reduce their memory needs and processing time considering an, as small as possible, number of samples to be generated since one can verify the tolerance at any time. This last improvement combined with the monotonicity property identified in SAN models with a component-wise order can
help us to solve rather huge models. 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 avoided for models with thousands of millions states. Only the statistical analysis of the generated samples remains to be dealt.

The perfect simulation of SAN allowing function integration during the sampling procedure is an alternative solution for models where iterative numerical algorithms are no longer possible with the current available resources.

ACKNOWLEDGMENTS

This work is partially funded by Petrobras (Conv. 0050.0048664.09.9). Afonso Sales receives grants from CAPES-Brazil (PNPD 02388/09-0). Paulo Fernandes is also funded by CNPq-Brazil (PQ 307284/2010-7). The order of authors is merely alphabetical.

REFERENCES