Transient Analysis of Superposed GSPNs

Peter Kemper
Informatik IV
Universität Dortmund
D-44221 Dortmund, Germany

Abstract

The paper considers transient analysis using randomization for superposed generalized stochastic Petri nets (GSPNs). Since state space explosion implies that space is the bottleneck for numerical analysis, superposed GSPNs profit from the structured representation known for its associated Markov chain. This moves the bottleneck for analysis from space for generator matrices to space for iteration vectors. Hence a variation of randomization is presented which allows to reduce space requirements for iteration vectors. An additional and welcome side effect is that during an initial phase, this algorithm avoids useless multiplications involving states with zero probability. Furthermore it accommodates to adaptive randomization in a natural way.

1 Introduction

Generalized stochastic Petri nets (GSPNs) are a modeling formalism for concurrent systems, whose mapping to its associated continuous time Markov chain (CTMC) is known for long [1]. If performance analysis aims at the transient behavior of a given GSPN, an ordinary differential equation (ODE) needs to be solved. Apart from ODE solvers, randomization is an established alternative [9, 10, 15, 18, 19]. Different variations exist to care for known limits and weaknesses, e.g. concerning stiff Markov chains [7] and adaptive randomization [5, 21] to reduce the number of iteration steps. However, for many models randomization is considered rather robust and efficient [18]. Randomization goes back to the work of Jensen [11], it is also called Jensen's method or uniformization.

As any state based numerical method for CTMCs, randomization suffers from the state space explosion problem which is frequently observed when the generator matrix Q of the associated CTMC is derived from a GSPN. If GSPNs are composed via fused transitions, the resulting GSPN is a so-called superposed GSPN (SGSPN) and the associated CTMC has a structured representation closely resembling the compositional structure at net level. SGSPNs and their structured representation originate from the work of Donatelli [6], who transferred results of Plateau and coworkers [16, 17] to stochastic Petri nets. A structured representation reliefs the impact of the state space explosion for numerical analysis as far as the space for matrix Q is concerned. However, it moves the bottleneck in terms of space from Q to the iteration vectors.

In this paper, we describe how randomization is combined with the structured representation known for SGSPNs [12]. Clearly, basic concepts directly carry over, however memory requirements are higher for randomization than for steady state analysis due to an additional vector. We present a new algorithm for a matrix vector multiplication, which focuses at a reduction of space requirements for iteration vectors. The algorithm is not restricted to transient analysis of SGSPNs by nature, but for this context it focuses on some crucial points: due to the structured representation of Q, iteration vectors dominate the space complexity; furthermore a degenerated initial distribution causes many states to retain a zero probability for a certain amount of iteration steps. The algorithm considers only states which are reachable within n steps for the computation of the n-th iteration vector. Therefore the algorithm is time-efficient in an initial phase of randomization. Nevertheless, its main focus point is to reduce space requirements by an alternative vector representation employing a stack. Grassmann [10] distinguishes between static and dynamic methods for randomization. A static method is based on a static representation of the generator matrix, which is computed before the iterative solution starts. In the dynamic method, the required parts of the generator matrix are generated on the fly in each iteration step and only for those states, which have a nonzero probability. The new algorithm compromises between these methods, since it uses a static, structured representation of the generator matrix, but enumerates dynamically those states which can have a nonzero probability in the n-th iteration step.

Furthermore we discuss randomization for unbounded SGSPNs in the context of structured representations. This is possible in principle, since the number of necessary iteration steps for randomization can be precalculated, which in turn limits the set of reachable states and results in a finite structured representation.

The paper is organized as follows: Section 2 defines superposed GSPNs and the structured representation for the associated Markov chain. Section 3 recalls the well known randomization method in the context of

1063-6714/97 $10.00 © 1997 IEEE

SGSPNs. In Section 4 we discuss transient analysis of unbounded SGSPNs using a structured representation. Section 5 contains a new variation of randomization following the reachability relation. Section 6 exercises a non-trivial example to demonstrate the benefits of the new algorithm and Section 8 summarizes the given results.

2 Superposed GSPNs and structured representations

We briefly introduce some basic notations and recall known results. The notation is taken from [12], we assume that the reader is familiar with GSPNs and their dynamic behavior [1, 3].

Definition 1 *A GSPN is an eight-tuple*

$$(P, T, \alpha, I, O, H, W, M_0)$$

where P is the set of places, T is the set of transitions such that $T \cap P = \emptyset$, $\alpha : T \to \{0, 1\}$ is the priority function, $I, O, H : T \to Bag(P)$, are the input, output, and inhibition functions, respectively, where $Bag(P)$ is the multiset on P, $W : T \to \mathbb{R}^+$ assigns a weight to each transition t with $\alpha(t) = 1$ and a rate to each transition t with $\alpha(t) = 0$, $M_0 : P \to \mathbb{N}_0$ is the initial marking: a function that assigns a nonnegative integer value to each place.

$T_E = \{t \in T | \alpha(t) = 0\}$ is the set of timed transitions, $T_I = T \backslash T_E$ is the set of immediate transitions. In a graphical representation, circles represent places, boxes (bars) represent timed (immediate) transitions. Arcs, leading from places to transitions, describe the input function. The set of input places for a transition t is ${}^\bullet t = \{p \in P | I(t)(p) \neq 0\}$. Function O is represented by arcs, leading from transitions to places, and $t^\bullet = \{p \in P | O(t)(p) \neq 0\}$ is the set of output places for $t \in T$. Arcs, denoting the inhibition function, are circle-headed, and ${}^\circ t = \{p \in P | H(t)(p) \neq 0\}$. An inhibitor arc (p, t) is labeled with the multiplicity of $H(t)(p)$, a value of 1 is usually omitted for readability. Arcs for functions I and O are labeled in the same manner. ${}^\bullet p = \{t \in T | O(t)(p) \neq 0\}$ is the set of transitions whose output bag contains place p. Analogously, we define $p^\bullet = \{t \in T | I(t)(p) \neq 0\}$, $p^\circ = \{t \in T | H(t)(p) \neq 0\}$. The notion can directly be extended to sets.

The dynamic behavior of a GSPN results from the firing of transitions yielding other markings $M : P \to \mathbb{N}_0$ than M_0. Immediate transitions have priority over timed transitions. An immediate transition t is enabled in a marking M (denoted by $M[t>$) iff $M \geq I(t)$ and $\forall p \in {}^\circ t : M(p) < H(t)(p)$. A timed transition t is enabled in a marking M iff no immediate transition is enabled in M, $M \geq I(t)$, and $\forall p \in {}^\circ t : M(p) < H(t)(p)$. Any transition $t \in T$ with $M[t>$ can fire, producing a new marking $M' = M - I(t) + O(t)$, denoted as $M[t> M'$. A sequence $M[t_1> M_1[t_2> M_2 \ldots M_{n-1}[t_n> M'$ is abbreviated as $M[\sigma> M'$ for a firing sequence $\sigma = t_1 t_2 \ldots t_n \in T^*$. An enabled timed transition t fires with a delay which is exponentially distributed with rate $W(t)$. In case of conflicts between immediate transitions in a marking M, an enabled immediate transition $\hat{t}$ fires with probability $W(\hat{t}) / \sum_{t \in M[\hat{t}>} W(t)$. Based on these definitions, the set of reachable markings/states (RS), the reachability graph (RG), the tangible reachability set (TRS), and the tangible reachability graph (TRG) can be defined as usual [3, 6].

For GSPNs, well-known techniques apply to derive a state transition matrix $\tilde{Q}$ from the TRG, such that the generator matrix Q of the underlying CTMC is given by $Q = \tilde{Q} - D$ with diagonal matrix D, where $D(i, j) = \sum_k \tilde{Q}(i, k)$ if $i = j$ and 0 otherwise.

Superposed GSPNs are GSPNs where, additionally, a partition of the set of places is defined, such that SGSPNs can be seen as a set of GSPNs which are synchronized by certain transitions.

Definition 2 *A SGSPN is a ten-tuple*

$$(P, T, \alpha, I, O, H, W, M_0, \Pi, TS)$$

where $(P, T, \alpha, I, O, H, W, M_0)$ is a GSPN, $\Pi = \{P^0, \ldots, P^{N-1}\}$ is a partition of P with index set $IS = \{0, \ldots, N-1\}$, $TS \subseteq T_E$ is the set of synchronized transitions. Moreover, Π induces a partition of transitions on $T \backslash TS$. A SGSPN contains N components $(P^i, T^i, \alpha^i, I^i, O^i, H^i, W^i, M_0^i)$ for $i \in IS$, with $T^i = {}^\bullet(P^i) \cup (P^i)^\bullet \cup (P^i)^\circ$ and $\alpha^i, I^i, O^i, H^i, W^i, M_0^i$ are functions α, I, O, H, W, M_0 restricted to P^i, resp. T^i. $IC(t) = \{i \in IS | t \in T^i\}$ is the set of involved components for $t \in T$.

This property is used to represent generator matrix Q for the CTMC underlying a SGSPN by a sum of tensor products defined on matrices which result from isolated components. We define tensor products according to [4], but only for square matrices, since only square matrices are relevant in our context.

Definition 3 *Tensor product, tensor sum*

Let $A^0, \ldots, A^{N-1}$ be square matrices of dimension $(k^i \times k^i)$ then their tensor product $A = \bigotimes_{i=0}^{N-1} A^i$ is defined by $a(x, y) = \prod_{i=0}^{N-1} a^i(x^i, y^i)$ where $x = \sum_{i=0}^{N-1} x^i g_i$ and $y = \sum_{i=0}^{N-1} y^i g_i$ with weights $g_0 = 1, g_i = k^{i-1} g_{i-1}$.

The tensor sum $B = \bigoplus_{i=0}^{N-1} A^i$ is then given by $\bigoplus_{i=0}^{N-1} A^i = \sum_{i=0}^{N-1} I_{l^i} \bigotimes A^i \bigotimes I_{r^i}$ where I_{l^i}, I_{r^i} are matrices of dimension $l^i \times l^i$, resp. $r^i \times r^i$ where $r^i = \prod_{j=0}^{i-1} k^j$, $l^i = \prod_{j=i+1}^{N-1} k^j$ and $I(a, b) = 1$ iff $a = b$ and 0 otherwise.

A tensor product formalizes the operation of multiplying every matrix element of one matrix with all matrix elements of the other matrices; these products of matrix elements are arranged in lexicographical order in the resulting matrix, for more details see, e.g., [20].

Any component i of a SGSPN is a GSPN. It can be analyzed in isolation yielding TRS^i and a generator matrix $Q^i = \bar{Q}^i - D^i$ as for any GSPN. A matrix $\bar{Q}^i$ does not contain vanishing markings any more, i.e., the elimination of vanishing markings was applied beforehand. Matrix $\bar{Q}^i$ can be seen as a sum of matrices $\bar{Q}^i = \sum_{t \in T^i} W(t)\bar{Q}^i_t$, such that nonzero entries are separated according to the timed transition t which contributes to that entry. The elimination of vanishing markings implies that $\bar{Q}^i_t(x,y)$ gives the conditional probability to reach marking M^i_y if transition t is fired in M^i_x. The terms of the sum which correspond to unsynchronized (local) timed transitions is denoted by $\bar{Q}^i_l = \sum_{t \in T^i \setminus TS} W(t)\bar{Q}^i_t$. We will regard only SGSPNs where the TRS^i of every component i is finite. A way to ensure this restriction by P-invariants is discussed in [12].

Theorem 1 *[12] The state-transition matrix $\bar{Q}$ for $PS = \times_{i=0}^{N-1} TRS^i$ of a SGSPN with finite TRS^i equals*

$$\bigoplus_{i=0}^{N-1} \bar{Q}^i_l + \sum_{t \in TS} W(t) \bigotimes_{i=0}^{N-1} \bar{Q}^i_t$$

where $\bar{Q}^i_t = I_{k^i}$ if $t \notin T^i$.

I_{k^i} is a $(TRS^i \times TRS^i)$ matrix with $I(a,b) = 1$ iff $a = b$ and 0 otherwise. The corresponding diagonal matrix D, such that the generator matrix is contained in $Q = \bar{Q} - D$, has a structured representation as well:

$$D = \bigoplus_{i=0}^{N-1} D^i_l + \sum_{t \in TS} W(t) \bigotimes_{i=0}^{N-1} D^i_t$$

where D^i_l, resp. D^i_t provide diagonal matrices with row sums of matrices $\bar{Q}^i_l$, resp. $\bar{Q}^i_t$. $PS = \times_{i=0}^{N-1} TRS^i$ denotes the product space obtained from the TRS^i of the components. The set of reachable tangible states TRS of a SGSPN is a subset of PS. A structured representation needs to be accomplished by additional permutation matrices [12] if $|TRS| << |PS|$, which frequently happens due to synchronized transitions.

Definition 4 *For a SGSPN with $|PS| = k$, a bijective function $perm : \{0, 1, \ldots, k-1\} \to \{0, 1, \ldots, k-1\}$ is a TRS-permutation if*

$\forall M_x \in PS : perm(x) < |TRS| \iff M_x \in TRS$
and
$\forall M_x, M_y \in TRS : perm(x) < perm(y) \iff x < y$

A TRS-permutation reorders states according to their reachability. It can be represented by a $(k \times k)$ matrix $\mathcal{P}$ with $\mathcal{P}(i,j) = 1$ if $j = perm(i)$ and 0 otherwise. $perm$ is bijective, so $\mathcal{P}^{-1}$ exists, and additionally $\mathcal{P}^{-1} = \mathcal{P}^T$. Let $x_p = x\mathcal{P}$ denote the permuted vector of a vector x, then the following transformation is applied to simplify the restriction of an arbitrary matrix vector multiplication to a submatrix given by TRS.

$$\begin{aligned} x^{(n+1)} = x^{(n)}H &\iff x_p^{(n+1)}\mathcal{P}^{-1} = x_p^{(n)}\mathcal{P}^{-1}H \\ &\iff x_p^{(n+1)} = x_p^{(n)}\mathcal{P}^T H\mathcal{P} \end{aligned}$$

The practical implications of employing a TRS-permutation are that iteration vectors $x_p^{(n)}, x_p^{(n+1)}$ can be represented by arrays of size $|TRS|$, where $perm(x) = i$ gives the appropriate position i for a state $M_x \in TRS$, as the probability of unreachable states is known to be zero. Furthermore, D_p^{-1} can be represented by an array of size $|TRS|$ in a similar way, and the same holds for $\mathcal{P}, \mathcal{P}^T$. As shown in [12], it is sufficient to use a single integer array of size $|TRS|$ to represent both, $\mathcal{P}$ and $\mathcal{P}^T$, for numerical analysis.

In this way, data structures for a matrix vector multiplication involving a structured representation can be limited in the size of $|TRS|$ instead of $|PS|$. If a matrix vector multiplication is performed by rows, it can be restricted to states in TRS, which are all located in the first part of $x_p^{(n)}$. So a TRS-permutation is an optional mean to account for $|TRS| << |PS|$ in the context of structured representations.

3 Randomization for structured representations

The goal of transient analysis is to compute the probability distribution $\pi(\tau)$ for a certain point of time τ for a given CTMC with initial distribution $\pi(0)$. The transient behavior of a CTMC is described by the Chapman/Kolmogorov ordinary differential equation (ODE) system

$$\dot{\pi}(\tau) = \pi(\tau)Q, \quad \pi(0) = \pi_0, \tag{1}$$

whose solution is described by

$$\pi(\tau) = \pi(0)e^{Q\tau}. \tag{2}$$

Apart from general methods to solve ODEs, randomization is a special method for transient analysis of CTMCs. It is also called uniformization [5] and Jensen's method [11]. It solves $\pi(\tau)$ by

$$\pi(\tau) = \pi(0) \sum_{n=0}^{\infty} e^{-\lambda\tau} \frac{(\lambda\tau)^n}{n!} P^n \tag{3}$$

with a stochastic matrix $P = I + 1/\lambda Q$ where $\lambda \geq max\{Q(i,i)\}, i \in TRS$. λ is called the uniformization rate.

(3) is usually truncated to the first k terms. k can be selected such that the resulting truncation error remains less than a given $\epsilon > 0$. Fox and Glynn [8] describe a method to compute a left and right truncation for given values of λ, τ, and ϵ. Since the left truncation just saves summation operations, but it does not

reduce the number of necessary matrix-vector multiplications, we therefore consider only right truncation in the following.

In principle, the method is simple, but as Grassmann already stated in [10], difficulties arise for an efficient and numerically stable implementation. Randomization is usually implemented by

$$\varphi^{(i)} = \begin{cases} \pi(0) & \text{if i = 0} \\ \varphi^{(i-1)}P & \text{otherwise} \end{cases} \quad (4)$$

$$f(i) = \begin{cases} e^{-\lambda\tau} & \text{if i = 0} \\ f(i-1)\frac{\lambda\tau}{i} & \text{otherwise} \end{cases} \quad (5)$$

$$x^{(i)} = \begin{cases} f(0)\varphi^{(0)} & \text{if i = 0} \\ x^{(i-1)} + f(i)\varphi^{(i)} & \text{otherwise} \end{cases} \quad (6)$$

where $x^{(k)} \approx \pi(\tau)$.

$f(i) = e^{-\lambda\tau}\frac{(\lambda\tau)^i}{i!}$ forms a Poisson distribution with parameter $\lambda\tau$; it is often referred to as a jump probability.

In terms of computational complexity, the crucial operation is the matrix vector multiplication to compute $\varphi^{(i)}$. Calculation of $\varphi^{(n)} = \pi(0)P^n$ is equivalent to the Power method for a CTMC with generator matrix Q, such that $P = I + \frac{1}{\lambda}Q$ for an appropriate value λ. $\varphi^{(i)}$ can be interpreted as the state probability of the CTMC after i state transitions.

So $x^{(k)}$ can be understood as the discrete probability vector after k jumps multiplied by the probability of having fired k timed transitions, summed over all possible number of jumps up to k. Since probabilities for very large numbers of jumps become very small, truncation of the summation to the first k terms results in a good approximation for sufficiently large k.

Randomization has often been considered the best method [10, 18] for computing transient state probabilities of CTMCs, but it suffers from a severe degradation in performance for increasing values of $\lambda\tau$. This is the case, whenever a highly dynamic model (λ gives the highest state departure rate) is analyzed for a relatively far time horizon τ. The problem appears in so-called "stiff" models, which can cause a large number of iteration steps and numerical instability. This problem is model dependent and beyond the scope of this paper; the interested reader is referred to e.g. [5, 7].

For large state spaces, (4) reveals similar problems as numerical methods for steady state analysis: namely the size of the matrix representation and the time used to perform a vector matrix multiplication are crucial for the applicability and performance of an iterative solution method. In consequence, considerations for steady state analysis of SGSPNs [12] are also valid for the context of randomization. For an implementation a conventional approach [10] uses a sparse matrix representation for P and three vectors of length $|TRS|$, namely $x^{(n)}$, $\varphi^{(n-1)}$, and $\varphi^{(n)}$. The structured representation of Q given in Theorem 1 can be used in this context as well, since $P = I + \frac{1}{\lambda}Q$, we obtain

$$P = D' + \frac{1}{\lambda}(\bigoplus_{i=0}^{N-1} \bar{Q}_l^i + \sum_{t\in TS} \bigotimes_{i=0}^{N-1} W(t)\bar{Q}_l^i)$$

where $D' = I - \frac{1}{\lambda}D$. A vector representation for diagonal entries D' is advisable compared to a structured representation in order to save repeated computations of diagonal values. The factor $\frac{1}{\lambda}$ can be easily integrated into tensor sum and product such that a computation of $\varphi^{(n)} = \varphi^{(n-1)}P$ can be performed by the same algorithms for a matrix-vector multiplication as in steady state analysis [20, 12]. This yields the following equations for the computation of $\varphi^{(n+1)}$:

$$\varphi^{(n+1)} = \varphi^{(n)}D' + \frac{1}{\lambda}\left[\varphi^{(n)}(\bigoplus_{i=0}^{N-1} \bar{Q}_l^i) + \sum_{t\in TS} \varphi^{(n)}(W(t) \bigotimes_{i=0}^{N-1} \bar{Q}_t^i)\right] \quad (7)$$

and in case of a TRS-permutation:

$$\varphi_p^{(n+1)} = \varphi_p^{(n)}D_p' + \frac{1}{\lambda}\left[\varphi_p^{(n)}\mathcal{P}^T(\bigoplus_{i=0}^{N-1} \bar{Q}_l^i)\mathcal{P} + \sum_{t\in TS} \varphi_p^{(n)}\mathcal{P}^T(W(t) \bigotimes_{i=0}^{N-1} \bar{Q}_t^i)\mathcal{P}\right] \quad (8)$$

The main difference to steady state analysis is that an additional vector for $x^{(n)}$ is required, which stores the weighted sum of vectors $\varphi^{(0)}, \varphi^{(1)}, \ldots, \varphi^{(n)}$.

4 Analysis of unbounded SGSPNs

Grassmann [10] already discussed randomization for infinite Markov chains, which is possible since truncation of (3) to k terms implies that one considers only the first k events, resp. firing of k timed transitions in Petri net terminology. Since a finite number of transitions firings only allows to reach a finite number of states, the computation of a probability distribution at a given time τ by randomization based on finite matrices is possible in principle. In [10], Grassmann describes a dynamic method, which consecutively generates sets of active states interpreting the model description; hashing techniques are used to decide whether a state is new or whether it has been reached before. In the following we describe an algorithm following a static approach and exploit the compositional structure of SGSPNs to handle unboundedness. The approach for SGSPNs heavily relies on an efficient TRS-exploration [13].

The essential idea is to consider the finite set of tangible states TRS_k which is reachable within k firings of timed transitions. The choice of an appropriate

value of k for randomization depends on the truncation error ϵ

$$1 - \sum_{n=0}^{k} e^{-\lambda\tau} \frac{(\lambda\tau)^n}{n!} < \epsilon \qquad (9)$$

for the truncation of the infinite sum in (3). For given values ϵ and τ, a corresponding k can be computed if λ is known. In GSPNs, uniformization rate λ is limited according to $\lambda \leq \lambda_0 = \sum_{t \in T_E} W(t)$. Since k increases with increasing values of $\lambda\tau$, this inequality is suitable to find an upper limit k_0 for the number of steps. We are interested in small values of k, so a minimal value of λ is desired.

Once k_0 is known, we can explore $TRS^i_{k_0}$ for each component i in isolation, where firing sequences are bound by k_0 (with respect to timed transitions). Finiteness is ensured by k_0, such that once the finite component state spaces have been computed, a finite, structured representation Q_0 is obtained.

$$Q_0 = \bigoplus_{i=0}^{N-1} \bar{Q}^i_l + \sum_{t \in TS} W(t) \bigotimes_{i=0}^{N-1} \bar{Q}^i_t - \bigoplus_{i=0}^{N-1} D^i_l - \sum_{t \in TS} W(t) \bigotimes_{i=0}^{N-1} D^i_t$$

Definition 2 implies, that synchronized transitions of an isolated component i are assumed to be enabled permanently as far as other components $j \neq i$ are concerned. So the firing sequences considered in the component state spaces are shortened projections of firing sequences in the complete model (if possible at all), such that the structured representation guarantees to contain TRS_{k_0} for the complete net.

The uniformization rate can be limited by $\lambda \leq \lambda_1 = \sum_{i=0}^{N-1} max_j D^i_l(j) + \sum_{t \in TS} W(t)$ at this point. Obviously $\lambda_1 \leq \lambda_0$ since $max_j D^i_l(j) \leq \sum_{t \in T^i \setminus TS} W(t)$. Since typically not all local timed transitions are enabled simultaneously, usually $\lambda_1 < \lambda_0$ and the corresponding k_1 is suitable to reduce component state spaces. A reduction of $TRS^i_{k_0}$ to $TRS^i_{k_1}$ is simplified, if states are indexed with respect to n-step reachability. A reduction of component state spaces reduces PS in the structured representation, which is helpful for the following TRS-exploration.

Based on component spaces $TRS^i_{k_1}$ and a reduced structured representation Q_1, we start a TRS-exploration using bitstate hashing with a perfect hash function [13]. The basic algorithm in [13] is easily tuned to limit reachability to k_1 transition firings. This yields TRS_{k_1}. A vector $\mu(j)$, $0 \leq j \leq k_1$, which gives the maximal departure rate $Q_1(i,i)$ observed within j steps, can be computed as a byproduct. Note that values in μ are non-decreasing. If $\mu(k_1) < \lambda_1$, we take $\mu(k_1)$ as a better estimate for λ in (9) and obtain a reduced value k_2. This leads to an iteration process if $\mu(k_2) < \mu(k_1)$. We iterate until a fixpoint $\mu(k_{i+1}) = \mu(k_i)$ is reached, which gives the desired appropriate minimal values λ and k of (9) for randomization. Component state spaces and corresponding matrices are reduced appropriately.

The resulting finite, structured representation of Q, resp. $P = I + \frac{1}{\lambda}Q$, can be used to proceed with randomization as discussed in the previous section. If space permits, diagonal values can be computed and stored once, otherwise they need to be recomputed according to (2) in each iteration step.

The described technique should not be overestimated in its ability to handle unbounded SGSPNs. Since unbounded places aggravate the state space explosion problem even within a limited number of steps, we expect that the above method is only useful for certain kinds of nets, where e.g. most components have finite TRS^i and relatively few components cause unboundedness.

5 The RS_n algorithm for randomization

In large CTMCs, the number of multiplications performed per iteration step clearly dominates the time complexity of randomization. Transient analysis of SGSPNs typically starts from an initial distribution $\pi(0)$, where only a few initial states – in fact in most cases just state M_0 – have a nonzero probability. In consequence, a certain number, n, of matrix vector multiplications is necessary, before a majority of states has a positive value in vector $x^{(n)}$, resp. $\varphi^{(n)}$. Obviously for $\varphi^{(n)}(i) = 0$ all multiplications $\varphi^{(n)}(i)P(i,.)$ can safely be omitted to increase efficiency.

We describe a variation of randomization which considers only those vector entries for the computation of $\varphi^{(n)}$ which are reachable within n jumps. Let $A_n = \{i \in TRS | \varphi^{(n)}(i) \neq 0\}$ be the set of active states at iteration step n. If an iteration algorithm is able to consider just elements of A_n and $|A_n| < |TRS|$, this algorithm is more efficient than a conventional matrix-vector multiplication $\varphi^{(n)} = \varphi^{(n-1)}P$. If the matrix-vector multiplication is performed by rows, a simple way to handle zero entries in $\varphi^{(n-1)}$ is to extend the conventional method by an appropriate test, i.e., to skip row i in the calculation of $\varphi^{(n)}$ if $\varphi^{(n-1)}(i) = 0$.

Sets of active states A_n develop over steps n in an irregular, model-dependent manner. Even $A_n \subseteq A_{n+1}$ is not true in general; for a counterexample consider a CTMC with $A_0 = \{M_0\}$ and $Q(M_0, M_0) = \lambda$, then $P(M_0, M_0) = 0$ and hence M_0 is not element of A_1. Note that this case is possible but not typical for applications. Since nonzero probabilities distribute from $\pi(0)$ following the reachability relation, the possibility of nonzero entries in $\varphi^{(n)}$ is closely related to reachability. Let $RS_n = \{M_i | M_0[\sigma > M_i \wedge |\sigma|_{T_E} \leq n\}$ be the set of markings which are reachable from M_0 with firing at most n timed transitions, i.e. $|\sigma|_{T_E}$ counts the number of timed transitions in firing sequence σ. Trivially, $RS_0 = \{M_0\}$. Sets RS_n show a number of

attractive properties, clearly $\forall n \in \mathbb{N} : RS_n \subseteq RS_{n+1}$ and $TRS = RS_\infty$. For any finite TRS, a fixed point $n \geq 0$ exists such that $RS_n = RS_{n+1}$. The following proposition clarifies the relation between nonzero entries in $x^{(n)}$ and elements of RS_n.

Proposition 1 *Let $x^{(n)}$ be defined as in (6), then $x^{(n)}(i) \neq 0 \iff M_i \in RS_n$.*

Proof. Note that $f(i) > 0$ by definition. The proposition follows directly from $x^{(n)} = \sum_{i=0}^{n} f(i)\pi(0)P^i$ and the fact that P gives the probabilities for one-step reachability between states in TRS, i.e., for $i \neq j$ it holds that $P(i,j) \neq 0 \iff \exists t \in T_E, \sigma' \in T_i^* : M_i[t\sigma' > M_j$.

Considering active states, at least $A_n \subseteq RS_n$ is true, or in terms of nonzero entries in $\varphi^{(n)}$, $\varphi^{(n)}(i) \neq 0 \Rightarrow M_i \in RS_n$. RS_n gives a set of states which can have a nonzero probability in $\varphi^{(n)}$, while states in $TRS \backslash RS_n$ must have zero entries in $\varphi^{(n)}$. In consequence, RS_n can be considered instead of A_n to obtain similar results, such that we aim for an algorithm which performs a vector-matrix multiplication only on rows corresponding to elements in RS_n.

In the context of large CTMCs, space is the main bottleneck for applications, hence we look for a memory efficient representation of RS_n. Obviously, $x^{(n)}$ implicitly represents RS_n, in the sense that it supports an efficient member operation for RS_n but it does not support an efficient enumeration of RS_n.

A natural way to enumerate RS_n is to start from RS_0 and to successively enumerate states of $RS_i \backslash RS_{i-1}$ for $0 < i \leq n$ by considering states which are directly reachable from states in RS_{i-1}. This procedure neatly agrees with the multiplications necessary to perform a vector-matrix multiplication with respect to RS_n. This leads to a calculation of $\varphi^{(n)}$ from $\varphi^{(n-1)}$ with the help of one additional stack:

Algorithm 1
Compute $\varphi^{(n)}$ for given $\varphi^{(n-1)}, x^{(n-1)}, P, M_0$

init: push(stack, M_0) ; $\forall i \in TRS : \varphi^{(n)}(i) = init$
while stack not empty
  M_i = *pop(stack)*
  for each $P(i,j) \neq 0$
    if $x^{(n-1)}(j) \neq 0 \wedge \varphi^{(n)}(j) = init$
    then push(stack, M_j)
    if $\varphi^{(n)}(j) = init$ *then* $\varphi^{(n)}(j) = 0$
    $\varphi^{(n)}(j) = \varphi^{(n)}(j) + \varphi^{(n-1)}(i)P(i,j)$
(*) $\varphi^{(n-1)}(i) = 0$

init $\notin [0,1]$ is a unique initial value for entries in $\varphi^{(n)}$, in order to distinguish it from any value obtained during iteration. The algorithm enumerates RS_{n-1} starting from $RS_0 = \{M_0\}$, since P provides information about the one step reachability and $x^{(n-1)}(j) \neq 0$ states that $M_j \in RS_{n-1}$. The additional condition $\varphi^{(n)}(j) = init$ ensures that each state of RS_{n-1} is pushed on the stack at most once.

The idea to multiply $\varphi^{(n-1)}P$ by rows is recommended in [10] in order to skip zero entries. It is also used for steady state analysis of SGSPNs with a structured representation and a TRS-permutation in order to restrict the iteration to reachable states [12]. Hence such a multiplication accommodates to randomization as well as to a structured representation.

Observe that the nonzero values of $\varphi^{(n-1)}$ are considered exactly once, such that $\varphi^{(n-1)}(i)$ can be reset in line (*) for reusing the space of $\varphi^{(n-1)}$ for vector $\varphi^{(n+1)}$ in the next iteration step. Additionally this allows to consider nonzero entries in $\varphi^{(n-1)}$ as the amount of work that has to be done in this iteration step. Note that a state M_j can be pushed on the stack only if a multiplication $\varphi^{(n-1)}(i)P(i,j)$ is added to an initial/zero entry in $\varphi^{(n)}(j)$, i.e. if $\varphi^{(n)}(j)$ receives a value for the first time. Therefore one can vary this procedure to get along with one iteration vector if the stack contains tuples $(M_i, \varphi^{(n-1)}(i))$. The idea is to overwrite $\varphi^{(n-1)}(j)$ with values contributing to $\varphi^{(n)}(j)$ and to evacuate an entry $\varphi^{(n-1)}(j)$ to the stack if necessary.

Mixing entries of $\varphi^{(n)}$ and $\varphi^{(n-1)}$ on the space of a single iteration vector φ is possible provided the algorithm is able to distinguish them. One way to do this is to use the sign bit of each entry. According to IEEE standard 754, a floating point has a special sign bit, which is unused by randomization, since entries in iteration vectors are nonnegative by definition. Inverting the sign bit does not influence the precision of an addition operation as long as the sign bits of both operands are equal.

No computation of $x^{(n)}$ takes place in Algorithm 1. Computation of $x^{(n)}$ requires $\varphi^{(n)}$. Since in a vector-matrix multiplication by rows it is difficult to decide when a value $\varphi^{(n)}(j)$ is complete, so computation of $x^{(n)}$ can only start after computation of $\varphi^{(n)}$ has terminated. Nevertheless, one can interleave computation of $x^{(n)}$ and $\varphi^{(n+1)}$, especially when $(M_j, \varphi^{(n)}(j))$ is pushed on the stack, one can safely use it to compute $x^{(n)}(j)$. In the following algorithm, computation of $\varphi^{(n+1)}$ is always one step ahead of the computation of $x^{(n)}$. Hence, two sign bits are used since computation of $\varphi^{(n+1)}$ requires RS_n, but the appropriate $x^{(n)}$ to characterize it is not available yet. So we use $RS_n = RS_{n-1} \cup A_n$, with $x^{(n-1)}(i) \neq 0 \iff M_i \in RS_{n-1}$ and $\varphi^{(n)}(i) \neq 0 \iff M_i \in A_n$.

The following algorithm uses two vectors x and φ and an additional stack containing tuples $(i, \varphi(i))$ to perform one step from $x^{(n-1)}$ to $x^{(n)}$ according to (6). Vectors x and φ are double precision vectors of lengths $|TRS|$. x contains initially $x^{(n-1)}$ and at termination $-x^{(n)}$. Computation of $\varphi^{(n+1)}$ is always one step

ahead of the computation of $x^{(n)}$, such that initially $\varphi = \varphi^{(n)}$ and at termination $\varphi = -\varphi^{(n+1)}$. In each step the signs of x and φ are inverted, but their absolute values coincide with values obtained in conventional randomization.

The sign bits of entries in φ and x are used as flags to denote whether the corresponding state has been pushed on the stack or not. The coding is as follows: if $\varphi(y) > 0 \vee x(y) > 0$ then $\varphi(y)$ contains value $\varphi^{(n)}(y)$; if $\varphi(y) \leq 0 \wedge x(y) \leq 0$ then $\varphi(y)$ contains value (or at least an intermediate result for) $\varphi^{(n+1)}(y)$.

For simplicity of notation, we consider just matrix P and not a structured representation of it. However, integration of a structured representation accommodates to a multiplication by rows and is straightforward. Let $A_0 = \{M_0\}$ and $f(n-1) > 0$ be given.

Algorithm 2 *n-th step in randomization,* $x = x^{(n-1)}, \varphi = \varphi^{(n)}$

for each $i \in A_0$:
  $push(stack, (i, \varphi(i)))$
  $x(i) = -x(i) - f(n-1)\varphi(i)$
  $\varphi(i) = 0$

while not empty stack
  $(i, v) = pop(stack)$
  for each $j : P(i,j) \neq 0$
    if $\varphi(j) > 0 \vee x(j) > 0$
    then $push(stack, (j, \varphi(j)))$
      $x(j) = -x(j) - f(n-1)\varphi(j)$
      $\varphi(j) = 0$
    $\varphi(j) = \varphi(j) - vP(i,j)$

Correctness follows from several observations:
1) No $push(stack, (j, \varphi(j)))$ with $\varphi(j) < 0$ occurs: Assume j is the first element pushed on the stack, due to $x(j) > 0$, but $\varphi(j) < 0$. Since initially $\varphi(j) \geq 0$, it must have been considered in line $\varphi(j) = \varphi(j) - vP(i,j)$ before, in order to obtain a negative value. However, to reach this line, the test $\varphi(j) > 0 \vee x(j) > 0$ must have been true beforehand, such that j would have been pushed on the stack already, which is a contradiction.
2) Each element j becomes at most once element of the stack: Since values v popped from the stack are nonnegative and $P(i,j) > 0$, for values $\varphi(j)$ holds: once $\varphi(j) \leq 0$ then always $\varphi(j) \leq 0$. Since only $\varphi(j) \geq 0$ are pushed on the stack, $f(n-1) \geq 0$, and initially $x(j) \geq 0$, we obtain for $x(j)$: after pushing j on the stack holds $x(j) \leq 0$ (and furthermore $\varphi(j) \leq 0$). Since $\varphi(j) \leq 0$ remains valid and $x(j)$ is changed only if $[\varphi(j) > 0 \vee x(j) > 0]$, $x(j) \leq 0$ remains valid also. Hence $\varphi(j) > 0 \vee x(j) > 0$ can only be initially true, and once j is pushed on the stack this condition remains invalid.
3) For each element i on the stack, all successor states j are considered and pushed on the stack, if j has not been pushed on the stack already, and if j was reached in the i steps before, which is obviously indicated by a positive probability $\varphi(j)$ (step n) or a positive value $x(j)$ (steps $0, \ldots, n-1$).

Let $nz(X)$ denote the number of nonzero entries in a subset X of rows in matrix P. For the computation of $\varphi^{(n+1)}$ the algorithm performs $nz(RS_n)$ multiplications and additions, $|RS_{n+1}|$ tests for zero and $|RS_n|$ push, pop operations. The computation of $x^{(n)}$ requires $|RS_n|$ additions and multiplications. In summary, calculation of $x^{(n)}$,$\varphi^{(n)}$ is in $O(nz(RS_n) + |RS_n|)$.

Clearly for a complete randomization algorithm either sign bits of x and φ can be reset after each step or the coding can be inverted, which is not described here. However, we refer to the complete algorithm as RS_n algorithm. Note that a transient set of states with zero probability can be recognized and handled by modifying set A_0, i.e., in each step, each $i \in A_0$ with $\varphi(i) = 0$ can be replaced by the set of its successor states. If such an i is then rereached again this is handled automatically in the same way as for all other states reached in this step.

For memory considerations, let d be the space for a double precision value, s the space for a state index, which is $s = \lceil log_2(|PS|) \rceil$ bits due to the mixed radix number representation. Let $sm(X)$ be the space for a sparse matrix X, then the space for the structured representation of Q, resp. P and the TRS-permutation is $m = \sum_{i=0}^{N-1} sm(\bar{Q}_l^i) + \sum_{t \in TS} \sum_{i \in IC(t)} sm(\bar{Q}_t^i) + (s+d)|TRS|$; alternatively a sparse matrix representation would use $m = sm(P)$. Let $0 \leq h \leq |TRS|$ be the maximal stack height observed during computation. The RS_n algorithm uses space $m + 2\,d\,|TRS| + h\,(s+d)$ due to vectors φ and x, and the stack, containing tuples $(i, \varphi(i))$. The stack need not increase towards $|TRS|$ for increasing values of n; it depends on the connectivity of the reachability graph and the order it is considered. The height of the stack also depends on the strategy, by which elements are taken from it; LIFO (Last in first out) implies that RS_n is explored Depth-First-Search (DFS). The stack contains all successor states of states in the current search path, which have not been explored yet. A recursive description of Algorithm 2 formally reduces the stack size to the current path but this does not pay off for practical purposes, since the recursion itself "stores" an equivalent amount of information and needs space for this as well. Alternatively to DFS also Breadth-First-Search can be considered by using a FIFO-queue instead of a stack. However, lower stack sizes are experienced for LIFO.

Ordinary randomization uses space $m + 3\,d\,|TRS|$ due to vectors $x^{(n-1)}$,x^n, φ. The RS_n algorithm is space efficient if $h\,(s+d) < d\,|TRS|$. If $d = 2s$ (a valid assumption for an implementation if $|PS| \leq 2^{32}$, 4 bytes per integer, 8 bytes per double precision value), it is efficient if $h < 2/3|TRS|$. The RS_n algorithm can exceed ordinary randomization at most by $s\,|TRS|$ as $h < |TRS|$. In [14] several examples are exercised revealing stack heights $0.15|TRS| \leq h \leq 0.65|TRS|$.

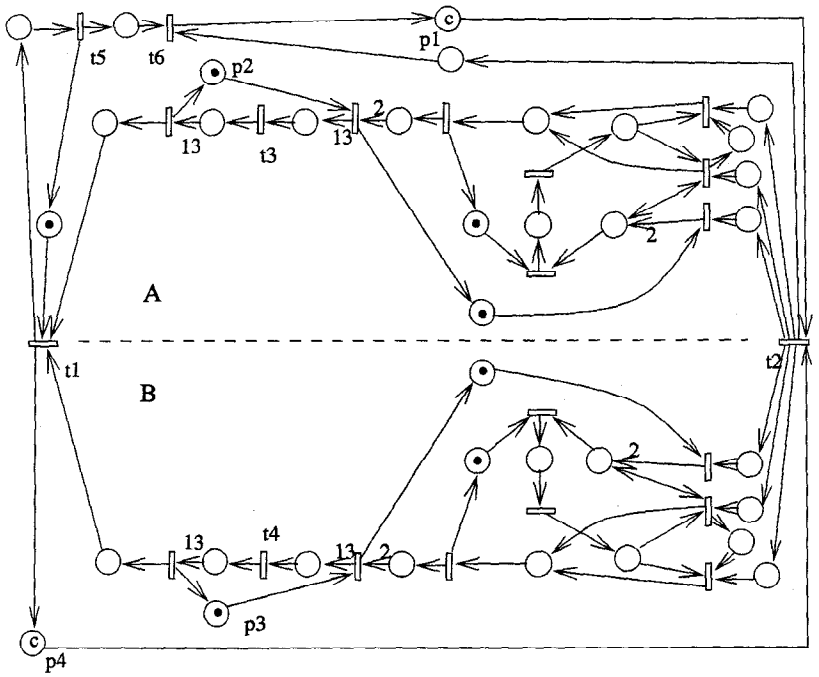

Figure 1: The example system

The RS_n-algorithm offers the chance to keep space requirements below the ordinary algorithm. Depending on A_0, this is at least fulfilled up to a certain step n. If this n is reached by randomization ($n \leq k$) the algorithm can fall back to the conventional algorithm. In this case, $|RS_n| \geq 2/3|TRS|$ which means that the advantage in efficiency of the RS_n algorithm is not very large anyway. In this sense we observe a 'graceful degradation' towards the conventional algorithm.

6 Analysis of an example SGSPN

Figure 1 shows a net similar to the benchprod model in [2]. It describes two production lines synchronized via transitions $t1$ and $t2$. The resulting product is fed back into the system via $t5$ and $t6$. Delays in production lines are often rather deterministic than exponentially distributed. Approximation of deterministic distributions by phase type distributions requires many phases, which is one cause of the state space explosion problem. Here, we model a deterministic delay in each production line by an Erlang_{13}-distribution, i.e. transitions $t3$ and $t4$ have to fire 13 times before the resource modeled by the token on place $p2$, resp. $p3$ is released. Since transitions $t1$ and $t2$ are timed, a partition as denoted by the dashed line in Fig. 1 is straightforward. Place $p4$ is redundant and represents an additional variable. $p4$ follows from a P-invariant; it is introduced to reduce PS for the given partition into components, for details of this technique see [12, 14].

The following results are obtained for parameter $c = 5$, which gives the initial marking for places $p1$ and $p4$. $|TRS| = 1,815,636$ and $|PS| = 8,351,136$, which means that about 21.7% of PS are reachable. Hence a structured representation needs to be supplemented by a TRS-permutation for an efficient solution. The structured representation consists of 3 matrices for each component, which altogether contain 22,290 nonzero entries, $|TRS^A| = 5,136$ and $|TRS^B| = 1,626$. Generating the structured representation and exploring TRS takes less than 40 s CPU-time/elapsed time by the improved algorithm given in [13]. This is negligible compared to the time used for matrix-vector multiplications. The generator matrix contains 8,918,528 nonzero entries. The fixed point $RS_n = RS_\infty$ is reached at $n = 256$. Figure 2 shows how the size of the stack, $|A_n|$, and $|RS_n|$ develop for an increasing number of steps n as observed by the RS_n-algorithm; all values are given relative to $|TRS|$. $RS_n \approx A_n$ and both converge towards TRS. The size of the stack is $h \leq 0.22\ |TRS|$ such that its space uses only about 33% of the space for an iteration vector $x^{(n)}$. The results are obtained on a Sparc 4 with 110 MHz CPU, 55 MB available primary memory (890 MB virtual). Clearly on this configuration, a conventional implementation with a sparse matrix collapses due to the well known memory thrashing effect. Figure 3 gives computation times (CPU and elapsed time) in seconds for the RS_n algorithm, indicated by RS_n,

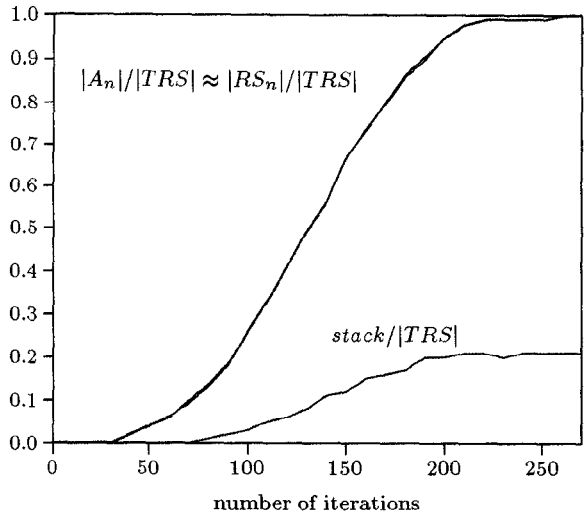

Figure 2: Height of stack, A_n, and RS_n relative to TRS during iteration

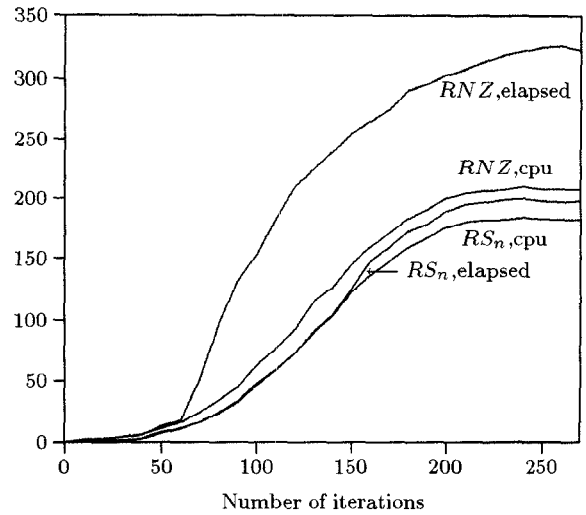

Figure 3: Computation times of RS_n and RZN algorithms in sec during iteration

and randomization using a structured representation with TRS-permutation and a test for nonzero vector entries, indicated by RNZ. Elapsed time and CPU-time for RNZ differ after about 60 iteration steps. This is due to the fact that probability mass in φ needs a number of iteration steps to distribute over TRS. This gives a certain locality in the beginning, which is lost once a sufficiently large set A_n has been obtained. Approaching towards memory limitations, elapsed time and CPU time differ due to time consuming paging operations. The RS_n algorithm is less demanding in terms of space, so a divergence of elapsed and CPU time is observed after about 140 iteration steps and the difference is rather mild compared to the RNZ algorithm. Results of Figure 3 indicate a clear advantage for the RS_n algorithm. However, this need not be the case in general. In [14], further example nets with other $|TRS|$/memory relations are exercised, where RS_n performs only initially faster, but takes more time per iteration step in the long run. In summary, RS_n is advisable in case of tight memory limitations and small stack heights. Since one can switch between both algorithms at each iteration step, it is presumedly best to have an implementation which selects an algorithm with respect to the observed stack height and available primary memory.

7 Integration of adaptive uniformization

The example above indicates that for large Markov chains the number of necessary iterations can be quite high and costly in terms of computation time. Adaptive randomization [21] aims at a reduction of the number of necessary iteration steps to compute $\pi(\tau)$. A large number of iterations can be caused be a large value λ which can be interpreted as a very fine discretization of the dynamic behavior. Due to the degenerated initial distribution, the departure rate corresponding to λ need not be experienced for a certain number of steps/jumps, such that a smaller rate would be appropriate as well. So the main idea in adaptive randomization is to adapt the uniformization rate λ_n at step n according to the maximum departure rate observed at states RS_n. This means that jump probabilities do not follow a Poisson distribution anymore but describe a general birth process, which has to be analyzed to calculate $f(i)$. Diener and Sanders discuss different variations of adaptive randomization in [5]. From a practical point of view and compared to standard randomization, adaptive randomization results in a different computation of function $f(i)$ and a variable uniformization rate λ_n. Since the latter can be trivially integrated into (7), resp. (8), and computation of function $f(i)$ is independent of the computation of $\varphi^{(i)}$, the new algorithm for randomization can be directly used with adaptive randomization. In fact, since our new algorithm follows reachability and the maximal diagonal value can be computed as a by-product during enumeration of RS_n, the new algorithm accommodates to adaptive randomization in a natural way.

8 Conclusions

In this paper, randomization is combined with the structured representation of CTMCs associated to SGSPNs. From steady state analysis of SGSPNs [12], it is already known that structured representations consider PS, the cross product of component state spaces instead of TRS, the tangible reachability set. We used the concept of TRS-permutation [12] to re-

strict randomization based on structured representations to TRS instead of PS.

A structured representation relies on a decomposition into components with finite state spaces. Nevertheless, we describe how unbounded SGSPNs can be analyzed by randomization in combination with a structured representation as well. The key observation is that for randomization, the number of iteration steps k to accomplish a required accuracy ϵ can be precalculated. In consequence, a structured representation can be restricted to the finite number of states, which are reachable within k steps. However, the applicability for this method is considered rather limited, due to the expected size of state spaces even for modest values of k in non-trivial, unbounded nets.

For bounded SGSPNs, a structured representation typically extends the size of solvable CTMCs by about one order of magnitude compared to conventional (sparse) matrix representations. Since space is the bottleneck for transient analysis of CTMCs and in the context of structured representations this bottleneck is caused by space for iteration vectors, the main contribution of this paper is the RS_n-algorithm. This algorithm aims at a reduction of space for iteration vectors. It performs randomization based on a structured representation but restricts itself to states which are reachable within n steps during computation of the n-th iteration vector. It accounts for the typically degenerated initial distribution $\pi(0)$ in transient analysis, where most states have zero probability. The algorithm is based on a new matrix vector multiplication which replaces one iteration vector of conventional randomization by a stack. It is space efficient if the stack height remains less than 2/3 of TRS. Since space is crucial in transient analysis, advantages of the RS_n algorithm for models whose solution reaches memory limitations are clear. A corresponding example is presented to demonstrate this effect. Finally it should be noted that the RS_n algorithm accommodates to adaptive randomization in a natural and straightforward way, since the RS_n algorithm enumerates all states reachable within n steps during computation of the n-th iteration step.

References

[1] M. Ajmone Marsan, G. Balbo, and G. Conte. *Performance Models of Multiprocessor Systems.* MIT Press, Cambridge, 1986.

[2] S. Caselli, G. Conte, and P. Marenzoni. Parallel state space exploration for GSPN models. In *Application and Theory of Petri Nets*, LNCS 935, pages 181–201. Springer, 1995.

[3] G. Chiola, M. Ajmone Marsan, G. Balbo, and G. Conte. Generalized stochastic Petri nets: a definition at the net level and its implications. *IEEE Trans. Software Engineering*, 19(2):89–107, Feb 1993.

[4] M. Davio. Kronecker products and shuffle algebra. *IEEE Transactions on Computers*, C-30(2):116–125, February 1981.

[5] J.D. Diener and W.H. Sanders. Empirical comparison of uniformization methods for continuous-time Markov chains. In W.J. Stewart, editor, *Computations with Markov Chains*, Kluwer Academic Publishers, 1995.

[6] S. Donatelli. Superposed generalized stochastic Petri nets: definition and efficient solution. In *Application and Theory of Petri nets 1994.* Springer, 1994.

[7] J. Dunkel and H. Stahl. On the transient analysis of stiff Markov chains. In *3rd IFIP Work. Conf. Dependable Computing for Critical Appl.*, 1992.

[8] B.L. Fox and P.W. Glynn. Computing poisson probabilities. *Comm. ACM*, 31:440–445, 1988.

[9] W.K. Grassmann. Transient solutions in Markovian queueing systems. *Computers & Operations Research*, (4):47–53, 1977.

[10] W.K. Grassmann. Finding transient solutions in markovian event systems through randomization. In W.J. Stewart, editor, *Numerical Solution of Markov Chains.* Marcel Dekker, 1991.

[11] A. Jensen. Markoff chains as an aid in the study of markoff processes. *Skand. Akuarietidskrift*, 36:87–91, 1953.

[12] P. Kemper. Numerical analysis of superposed GSPNs. *IEEE Trans. on Software Engineering*, 22(9), Sep 1996.

[13] P. Kemper. Reachability analysis based on structured representations. In *Application and Theory of Petri Nets*, LNCS 1091. Springer, 1996.

[14] P. Kemper. *Superposition of generalized stochastic Petri nets and its impact on performance analysis.* PhD thesis, Universität Dortmund, Krehl-Verlag, 1996 (in English).

[15] R.A. Marie, A.L. Reibman, and K.S. Trivedi. Transient analysis of acyclic Markov chains. *Performance evaluation*, (7):175–194, 1987.

[16] B. Plateau. On the stochastic structure of parallelism and synchronization models for distributed algorithms. In *ACM Sigmetrics Conf. Measurement and Modelling of Comp. Sys.* 1985.

[17] B. Plateau and K. Atif. Stochastic automata network for modelling parallel systems. *IEEE Trans. on Software Engineering*, 17(10):1093–1108, 1991.

[18] A. Reibman and K.S. Trivedi. Numerical transient analysis of Markov models. *Comput. Opns Res.*, 15:19–36, 1988.

[19] A.L. Reibman, R. Smith, and K.S. Trivedi. Markov and Markov reward model transient analysis: and overview of numerical approaches. *Europ. J. Operational Research*, (40):257-267, 1989.

[20] W.J. Stewart. *Introduction to the numerical solution of Markov chains.* Princeton University Press, 1994.

[21] A.P.A. van Moorsel and W.H. Sanders. Adaptive uniformization. *Stochastic Models*, 10(3), 1994.