Numerical Analysis of Superposed GSPNs

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

Abstract

The numerical analysis of various modeling formalisms profits from a structured representation for the generator matrix Q of the underlying continuous time Markov chain, where Q is described by a sum of tensor (Kronecker) products of much smaller matrices. In this paper we describe such a representation for the class of superposed generalized stochastic Petri nets (SGSPNs), which is less restrictive than in previous work. Furthermore a new iterative analysis algorithm is proposed. It pays special attention to a memory efficient representation of iteration vectors as well as to a memory efficient structured representation of Q. In consequence the new algorithm is able to solve models which have state spaces with several millions of states, where other exact numerical methods become impracticable on a common workstation.

1 Introduction

Generalized stochastic Petri nets (GSPNs) [1, 8] provide a concise and powerful method for the specification and analysis of complex dynamic systems. Their mapping to a continuous time Markov chain (CTMC) and subsequent analysis for transient and steady state distributions to derive performance measures has been known for long, and a rich variety of software tools exists which employ this (conventional) numerical analysis, among others [4, 7, 11, 19]. The conventional method follows basically 3 steps

1. state space exploration, which yields the reachability graph (RG),
2. elimination of vanishing markings, which reduces RG to the tangible reachability graph (TRG) and yields the generator matrix Q,
3. application of a numerical iteration scheme.

Due to the well known state space explosion problem, numerical analysis is cumbersome for large models where the underlying CTMC contains more than 10^6 states. In these cases the conventional method frequently collapses for a given computer configuration due to lack of primary memory. This takes place even if sparse matrix structures are employed for the representation of the generator matrix Q of the CTMC.

If the CTMC shows certain regularities, which are typically imposed by the modeling formalism it is derived from, then Q can be described by a set of much smaller matrices, which are combined via tensor operations [2, 13]. This memory efficient representation of Q usually increases the size of solvable CTMCs by one order of magnitude. Such structured representations are known for stochastic automata networks (SANs) [22, 23], for certain hierarchical colored stochastic Petri nets [5], and for superposed generalized stochastic Petri nets (SGSPNs) [15], which are an extension of superposed stochastic automata [14]. In SGSPNs, the idea is to combine a set of originally independent GSPNs into a single superposed GSPN by synchronization of transitions. At net level, the synchronization takes place by merging of transitions; the resulting transitions are then called synchronized transitions. This concept is closely related to the concept of SANs and Markovian process algebras [17, 16], which also consider synchronized actions. However, in this paper, we focus on SGSPNs.

Formally, SGSPNs are GSPNs for which a modeler provides a partition into components. Clearly, a partition which yields advantages for numerical analysis does not necessarily exist for an arbitrary GSPN. The problem of finding such a partition is beyond the scope of this paper. However, if a model is created in a modular way, and if interaction between modules/components is synchronous, a partition into components results quite naturally from the modeling process.

We consider analysis techniques for a given SGSPN. In SGSPNs, the isolated components are GSPNs themselves, such that the TRG of a single component can be computed by following the conventional approach. The superposition of components into a SGSPN can be used to compose the TRGs of these components into a structured representation of Q for the SGSPN. This is possible under the following restrictions [15]: 1. all synchronized transitions are timed, 2. the TRGs of the isolated components are strongly connected, and 3. firing of a synchronized transition leads to a tangible state. In this case, Q is described by a sum of tensor products which consist of matrices, derived from the TRGs of the isolated components. The 2nd restriction formulates a necessary condition for ergodicity of the associated CTMC. The 3rd restriction results from the formal derivation of the structured representation in [15] and is rather awkward from a modeler's point of view. In this paper, we give a similar structured representation of Q, but follow a different argumentation, such that we can drop the 3rd restriction. In our description of Q, the firing of synchronized transitions may lead to vanishing or tangible states. This extends the set of SGSPNs for which a concise[1] structured description of Q is known, and hence improves efficiency of numerical analysis methods based on it.

The main advantage of a structured representation is that it is a very memory efficient matrix representation, which is essential for the applicability of any numerical analysis. In case of SGSPNs, the price paid for this advantage is that the cross product of state spaces of isolated components – in the following denoted by $\mathcal{PS}$ for product space – is considered to be the tangible reachability set ($\mathcal{TRS}$) of the SGSPN, although often $|\mathcal{PS}| >> |\mathcal{TRS}|$ due to the restrictions imposed by synchronization. In the context of structured descriptions, the fact that introducing additional restrictions might exclude reachability of certain states in $\mathcal{PS}$ has been mentioned before in [12, 15].

[1] in terms of the number of terms in the sum of tensor products of a structured description

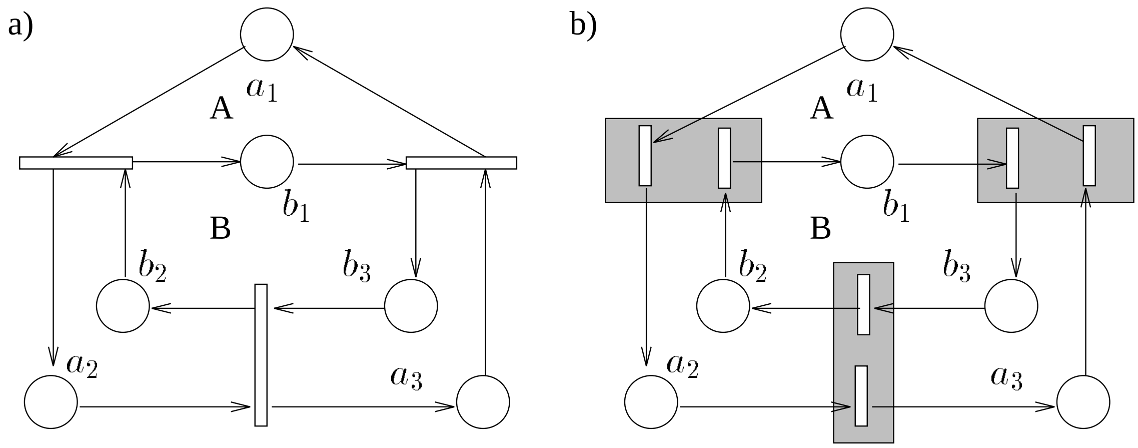

Figure 1: a) SGSPN resulting from b) by merging shaded transitions of GSPNs A and B

The following simple example demonstrates the impact of synchronization on the relation between $\mathcal{TRS}$ and $\mathcal{PS}$. Consider the net in Fig. 1 b), which consists of two independent subnets A and B, such that its $\mathcal{TRS}$ is simply the cross product of the tangible reachability sets of its subnets. The net in Fig. 1 a) is a SGSPN with two components A and B. It results from the net in Fig. 1 b) by merging the transitions in each shaded box into a single transition, a then so-called synchronized transition. We want to compare the $\mathcal{TRS}$ of the SGSPN with the cross product of component state spaces. Let $M_0 = (a_1 = a_2 = 0, a_3 = p, b_1 = b_2 = p + m, b_3 = 0)$ be an initial marking where p and m are non-negative parameters. By combinatorial arguments, it is easily obtained that

$$|\mathcal{PS}| = |\mathcal{TRS}^A||\mathcal{TRS}^B| = \begin{pmatrix} p+2 \\ p \end{pmatrix} \begin{pmatrix} 2(p+m)+2 \\ 2(p+m) \end{pmatrix} \geq \begin{pmatrix} p+2 \\ p \end{pmatrix} = |\mathcal{TRS}| \tag{1}$$

since only tokens on a_1, a_2, a_3 distribute freely over places a_1, a_2, a_3 and places b_1, b_2, b_3 are redundant for the given M_0 in the SGSPN. Due to the additional degree of freedom gained from parameter m, initial markings can be chosen to increase $\mathcal{PS}$ arbitrarily compared to $\mathcal{TRS}$. Synchronization restricts the behavior of the SGSPN drastically compared to the behavior of its independent components in Fig. 1 b). The example is surely a worst case example, but it clearly indicates that a numerical analysis method for SGSPNs based on a structured representation of Q must take care of this problem, where Q is a $(|\mathcal{PS}| \times |\mathcal{PS}|)$ matrix. Otherwise the method becomes hopelessly inefficient for SGSPNs of this type.

This problem is attacked and solved in this paper. We present a numerical analysis method which employs a structured representation of Q, restricts the size of iteration vectors to the size of $\mathcal{TRS}$, and considers only states of $\mathcal{TRS}$ in its iteration scheme. For example, the new analysis algorithm is able to analyze a SGSPN with $|\mathcal{PS}| \approx 56 \cdot 10^6$ and $|\mathcal{TRS}| \approx 1.6 \cdot 10^6$ on a Sparc station with 64 MB primary memory, as described in Sec. 6.

Section 2 provides the theoretical basis for subsequently presented algorithms; we define the modeling formalism (GSPNs and superposed GSPNs) and describe a structured representation of the generator matrix Q of the associated CTMC. A structured representation employs tensor operations, which are defined as well. Once a structured representation is established, subsequent sections consider ways to use it efficiently in numerical analysis. The general problem is that often $|\mathcal{PS}| >> \mathcal{TRS}$, as indicated by the example above. We follow two orthogonal approaches: in Sec. 3, we show how P-invariants can be used to avoid unused states in component state spaces. This approach aims for a reduction of $|\mathcal{PS}|$. However, it is not sufficient to avoid all unreachable states. In Sec. 4, we follow an orthogonal approach: we do not work on the cause but on the negative effect of unreachable states for the efficiency of numerical analysis algorithms. By the help of a permutation, we distinguish reachable from unreachable states, such that an iterative numerical algorithm can focus on $\mathcal{TRS}$. An algorithm is presented which computes $\mathcal{TRS}$ and the required permutation. In Sec. 5, we compose the results obtained so far to a new iterative numerical algorithm, which uses a structured representation, but restricts itself to $\mathcal{TRS}$ if $|\mathcal{PS}| >> |\mathcal{TRS}|$. In Sec. 6, we exercise a GSPN model of a courier protocol software taken from literature to demonstrate applicability and limits of our approach.

2 Definitions and theoretical basis

The notation for SGSPNs mainly follows [15]. We briefly introduce some basic notations and assume that the reader is familiar with GSPNs and their dynamic behavior [1, 8].

Definition 1 *A GSPN is an eight-tuple*

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

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

Let $T_E = \{t \in T | \pi(t) = 0\}$ be the set of timed transitions and $T_I = T \backslash T_E$ be the set of immediate transitions. In a graphical representation, places are shown as circles, timed transitions as boxes, and immediate transitions as black bars. Arcs, leading from places to transitions, describe the input function, and ${}^\bullet t = \{p \in P | I(t)(p) \neq 0\}$ gives the set of input places for a transition t. 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.

For finite sets P and T, functions I, O, H, and M_0 can be described by matrices, resp. vectors as well. The incidence matrix C is a $(P \times T)$ matrix, $C(p,t) = -I(t)(p) + O(t)(p)$. GSPNs show a dynamic behavior, such that other markings $M : P \rightarrow \mathbb{N}_0$ apart from M_0 can be obtained. M denotes a marking for all places, and $M(p)$ gives the marking of a certain place p. The dynamic behavior results from the firing of transitions.

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 $M' = M - I(t) + O(t)$, which is denoted by $M[t> M'$. 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 t fires with probability $W(t) / \sum_{t \in M[t>} W(t)$. Based on these definitions, the set of reachable markings/states (RS), the reachability graph (RG), the tangible reachability set ($\mathcal{TRS}$), and the tangible reachability graph (TRG) can be defined as usual [8, 15].

For GSPNs, well-known techniques apply to derive a state transition matrix $\bar{Q}$ from the TRG, such that the generator matrix Q of the underlying CTMC is given by $Q = \bar{Q} - D$ with diagonal matrix D

$$D(i,j) = \begin{cases} \sum_k \bar{Q}(i,k) & \text{if } i = j \\ 0 & otherwise \end{cases}$$

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*

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

where
$(P, T, \pi, I, O, H, W, M_0)$ *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* $T \backslash TS$. *Such a SGSPN contains* N *components*

$$(P^i, T^i, \pi^i, I^i, O^i, H^i, W^i, M_0^i)$$

for $i \in IS$, *where* $T^i = {}^\bullet(P^i) \cup (P^i)^\bullet \cup (P^i)^\circ$ *and* $\pi^i, I^i, O^i, H^i, W^i, M_0^i$ *are the functions* π, 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* $t \in T$.

Figure 1 a) shows a SGSPN, which contains two components A and B; Fig. 1 b) shows components A and B in isolation - the shaded boxes are used to indicate which transitions are merged to obtain the SGSPN.

Note that synchronized transitions of a SGSPN are timed by definition. This ensures that, within a SGSPN, components have "borderlines" just built from timed transitions. Consequently, the firing of immediate transitions in different components is rather independent in that the firing of an immediate transition in component i cannot enable or disable the firing of an immediate transition in component $j \neq i$ within the SGSPN. This is a nice special case of [3]: immediate transitions in different components cannot be in the same equal conflict set (ECS), i.e., they cannot enable or disable each other. This simplifies the elimination of vanishing markings considerably. According to [3], the state transition matrix $\bar{Q}$ of the total (S)GSPN can be calculated from

$$\bar{Q}(i,j) = \sum_{t \in T_E : \exists \sigma \in T_I^* : M_i[t\sigma > M_j} W(t) \prod_{n=0}^{N-1} Prob[F^n(t, M_i) \rightarrow M_j^n] \tag{2}$$

$W(t)$ denotes the firing rate of timed transition t. M_j^n is the restriction of the tangible marking M_j to component n. $F^n(t, M_i)$ denotes the successor marking after firing transition t in marking M_i with restriction to component n. $F^n(t, M_i)$ can be the start of a firing sequence σ^n of immediate transitions which is local in n and which leads to the tangible state M_j^n, so $Prob[F^n(t_k, M_i) \rightarrow M_j^n]$ gives the probability to reach M_j^n over all such sequences σ^n, which can be empty, under the condition that t is fired in M_i^n. Informally, the idea is that the probability of firing sequences of immediate transitions, enabled by firing of a timed transition t, is given by a product of subsequences, where each subsequence consists of transitions of a single ECS. Since transitions in different ECS do not interfere, their firing probabilities only require a normalization which is local to the ECS. A partition into ECSs is naturally given in SGSPNs by the partition into components, such that the product in (2) considers all N components in an arbitrary but fixed order, which is equivalent to the introduction of additional priorities on different ECS in [3].

We will make use of this property in proving that a representation of Q for the CTMC underlying a SGSPN is correct, which represents Q by a sum of tensor products and allows that firing of a synchronized transition enables immediate transitions. The definition of the tensor (Kronecker) product is based on a mapping function, using a mixed radix number representation [13].

Definition 3 *Mapping function mix*
Let $TRS^i = \{0, 1, \ldots, k^i - 1\}$ be some finite sets with arbitrary but fixed constants k^i for all $i \in \{0, 1, \ldots, N-1\}$ and $k = \prod_{i=0}^{N-1} k^i$.
A mapping $mix : \times_i TRS^i \longrightarrow \{0, 1, \ldots, k-1\}$ is defined by

$$mix(x^{N-1}, \ldots, x^1, x^0) = \sum_{i=0}^{N-1} x^i g_i$$

with weights $g_0 = 1, g_i = k^{i-1} g_{i-1}$.

A vector $(x^{N-1}, \ldots, x^0) \in \times_i TRS^i$ is the mixed radix number representation of $x = mix(x^{N-1}, \ldots, x^0)$ with respect to basis $(k^{N-1}, \ldots, k^0)$. In the following definition of tensor product, we follow the notation in [13], but regard only the restricted case of square matrices to keep a concise notation, as only square matrices occur in the context of SGSPNs.

Definition 4 *Tensor product and sum for square matrices*
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 $x = mix(x^{N-1}, \ldots, x^0)$ and $y = mix(y^{N-1}, \ldots, y^0)$.

The tensor sum $B = \bigoplus_{i=0}^{N-1} A^i$ is then given by $\bigoplus_{i=0}^{N-1} A^i = \sum_{i=0}^{N-1} I_{l^i} \otimes A^i \otimes I_{r^i}$ 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., [24].

The components of a SGSPN are GSPNs themselves. They can be analyzed in isolation to obtain corresponding tangible reachability sets TRS^i as for any GSPN. In the following, we will regard only SGSPNs where the TRS^i of every component i is finite. Due to our assumption that all states in TRS^i are consecutively numbered from 0 to $k^i - 1$, mix induces such a numbering on $\times_{i=0}^{N-1} TRS^i$ as well. Since this numbering allows to identify states, we will not distinguish between a state $M_x = (M_x^{N-1}, \ldots, M_x^0)$ and its number, resp. component numbers, $x = mix(x^{N-1}, \ldots, x^0)$ in order to preserve readability. Furthermore, let $\mathcal{PS} = \times_{i=0}^{N-1} TRS^i$ denote the product space obtained from the TRS^i of the components. The set of reachable tangible states of a SGSPN is denoted by $\mathcal{TRS}$. Since the component state spaces TRS^i for $\mathcal{PS}$ are obtained from isolated components, the enabling conditions of transitions are less restrictive than the enabling conditions of synchronized transitions in the SGSPN. In consequence, any firing sequence observed in the SGSPN can be imitated by a firing sequence in the set of components, such that we obtain the following property.

Proposition 1 *All $x \in \mathcal{TRS}$ are also element of $\mathcal{PS}$.*

Together with TRS^i, state transition matrices $\bar{Q}^i$ can be derived for every isolated component i, such that a $\bar{Q}^i$ 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, or, more formally, $Prob[F^i(t, M^i_x) \rightarrow M^i_y]$ like in (2).

The terms of the sum which correspond to unsynchronized (local) timed transitions shall be denoted by $\bar{Q}^i_l = \sum_{t \in T^i \setminus TS} W(t) \bar{Q}^i_t$.

Theorem 1 *The state-transition matrix $\bar{Q}$ for $\mathcal{PS} = \times_{i=0}^{N-1} TRS^i$ 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 $\bar{Q}^i_t = I_i$ if $t \notin T^i$.

I_i is a $(\mathcal{TRS}^i \times \mathcal{TRS}^i)$ matrix with $I(a,b) = 1$ iff $a = b$ and 0 otherwise; the index for component i is a subscript to avoid confusion with the input function of this component. The main ideas underlying the following proof are that firings of local transitions are well described by a tensor sum of local state transition matrices

$\bar{Q}^i_l$ – this is well known, for a detailed discussion see, e.g., [24] – and that firing of a synchronized transition followed by a firing sequence of immediate transitions in various components profits from the fact that immediate transitions in different components belong to different ECSs and cannot disable each other. The latter observation allows to exploit (2) for the calculation of a matrix entry, which happens to coincide with the matrix entries that result from $\sum_{t\in TS} W(t) \bigotimes_{i=0}^{N-1} \bar{Q}^i_t$, as shown below.

Proof. of Theorem 1

Regard two markings $M_x, M_y \in \mathcal{PS}$, with $x = mix(x^{N-1}, \ldots, x^0)$ and $y = mix(y^{N-1}, \ldots, y^0)$. Let $\bar{Q} = \sum_{t\in TS} \bar{Q}_t + \sum_{e=0}^{N-1} \bar{Q}_{l^e}$. This notation distinguishes the cause for a nonzero matrix entry. This is necessary, because the value on a matrix position $\bar{Q}(x,y)$ can be a sum of values caused by the possible firing of several timed transitions, performing the same transformation of marking M_x into marking M_y. So $\bar{Q}_t$ is a $(\mathcal{PS} \times \mathcal{PS})$ matrix, containing all nonzero entries caused by firing a synchronized transition t, whereas $\bar{Q}_{l^e}$ contains all nonzero entries caused by firing a timed transition of $T^e \backslash TS$ of a component e.

The proposed matrix representation can be transformed. According to Def. 4
$\bigoplus_{i=0}^{N-1} \bar{Q}^i_l + \sum_{t\in TS} W(t) \bigotimes_{i=0}^{N-1} \bar{Q}^i_t = \sum_{i=0}^{N-1} I_{l^i} \otimes \bar{Q}^i_l \otimes I_{r^i} + \sum_{t\in TS} W(t) \bigotimes_{i=0}^{N-1} \bar{Q}^i_t$.

In order to prove the theorem, we show that

1. for any component e : $\bar{Q}_{l^e} = I_{l^e} \otimes \bar{Q}^e_l \otimes I_{r^e}$

By definition, the enabling condition and the firing rate of any local timed transition in e is independent of the marking in other components. Furthermore, all synchronized transitions are timed. Hence, immediate transitions enabled by the firing of a local timed transition in e must be local in e as well, and no immediate transitions in other components can be enabled. In consequence, the value of any nonzero entry $\bar{Q}_{l^e}(x,y) = \bar{Q}^e_l(x^e, y^e)$. Since only local transitions of e are involved, the marking in other components remains obviously unchanged. In summary, we have

$$\bar{Q}_{l^e}(x,y) = \begin{cases} \bar{Q}^e_l(x^e, y^e) & \text{if } M^i_x = M^i_y \text{ for all } i \neq e \\ 0 & \text{otherwise} \end{cases}$$

Since matrices I are 1 for all $I(a,b)$ where $a = b$ and 0 otherwise, and a product is nonzero iff all of its factors are nonzero, $\bar{Q}_{l^e}(x,y) = [\prod_{i=e+1}^{N-1} I_i(x^i, y^i)]\ \bar{Q}^e_l(x^e, y^e)$ $\prod_{i=0}^{e-1} I_i(x^i, y^i)$. So, owing to Def. 4, $\bar{Q}_{l^e} = I_{l^e} \otimes \bar{Q}^e_l \otimes I_{r^e}$.

It remains to show that

2. for any $t \in TS$: $\bar{Q}_t = W(t) \bigotimes_{i=0}^{N-1} \bar{Q}^i_t$

A synchronized transition t is enabled in state $M_x \in \mathcal{PS}$ iff t is enabled in all components e with $t \in T^e$. (Note that the initial marking M_0 with $M = N = 1$ in Fig. 2 but $M_0(p33) = 0$ and $M_0(p34) = 1$ can serve as an example for M_x and $t = t28$ to illustrate the argumentation.) Firing of t can only change the marking of places in components e with $t \in T^e$, but the marking of places in other components remains unchanged. Since M_x is a tangible state, no immediate transition can be enabled at M_x. Furthermore, all immediate transitions enabled after firing t can again only be transitions in components e with $t \in T^e$. Since the borderline between components is formed by timed transitions, the immediate transitions which are enabled by firing t and which belong to different components belong to different ECSs. According to (2): $\bar{Q}_t(x,y) = W(t) \prod_{i=0}^{N-1} P[F^i(t, M_x) \rightarrow M^i_y]$. By definition of $\bigotimes_{i=0}^{N-1} \bar{Q}^i_t$, each nonzero element $\bar{Q}^i_t(x^i, y^i)$ gives $P[F^i(t, M^i_x) \rightarrow M^i_y] = P[F^i(t, M_x) \rightarrow M^i_y]$ iff t is enabled in M_x and $t \in T^i$, according to the independence of local immediate transitions. Once again, the product $W(t) \prod \bar{Q}^i_t(x^i, y^i)$ ensures that the resulting value is nonzero iff all factors are nonzero, i.e., iff all components i with $t \in T^i$ fulfill $\bar{Q}^i_t(x^i, y^i) = P[F^i(t, M^i_x) \rightarrow M^i_y] \neq 0$ and all components i with $t \notin T^i$ $\bar{Q}^i_t((x^i, y^i) = 1$, which, in turn, is only the case if $M^i_x = M^i_y$, since for $t \notin T^i$: $\bar{Q}^i_t = I_i$ is 1 for all $I_i(x^i, y^i)$ where $x^i = y^i$ and 0 otherwise. □

Starting from a structured description of state transition matrix $\bar{Q}$, it is straightforward to derive the diagonal matrix D, such that the generator matrix is given by $Q = \bar{Q} - D$. In the following, we will not attempt to obtain a structured description of D and rather use a standard representation, a vector enumerating the diagonal values of D. So far, we gave a representation of Q based on a structured description of $\bar{Q}$, which can directly be employed within numerical analysis methods. E.g., a Jacobi iteration can be performed by $x^{(n+1)} = x^{(n)}\bar{Q}D^{-1}$. The main idea is that the vector-matrix multiplication $x^{(n)}\bar{Q}$ can be performed by multiplying appropriate projections of $x^{(n)}$ with matrix elements of $\bar{Q}^i_l$, resp. $\bar{Q}^i_t$, cf. [24, 25]. In the following, we come back to the problem stated in Sec. 1, namely that often $|PS| >> |TRS|$. We describe how to reduce $|PS|$ and how to avoid overhead imposed by unreachable states in $\mathcal{PS}$.

3 Upper limits derived from SGSPN

In this section, we consider the relationship between P-invariants of a SGSPN and P-invariants of its components. An integer vector $x \in \mathbb{Z}^{|P|}$ is a P-invariant if $xC = 0$ where C denotes the incidence matrix. A P-invariant x is semi-positive if $x \in \mathbb{N}_0^{|P|}$. The set of places with nonzero entries in x are called the support of x. Since the components of a SGSPN are superposed by synchronization of transitions, superposition of GSPNs into a SGSPN preserves the P-invariants of involved GSPNs. More formally, let $x_i \in \mathbb{N}_0^{|P^i|}$ be a semi-positive P-invariant of an isolated component i. The corresponding vector $X_i \in \mathbb{N}_0^{|P|}$ in the SGSPN is then given by

$$X_i(p) = \begin{cases} x_i(p) & \text{if } p \in P^i \\ 0 & \text{otherwise} \end{cases}$$

Lemma 1 *Let X_i be the corresponding vector in a SGSPN for a semi-positive P-invariant x_i of a component i, then $X_iC = 0$, i.e., X_i is a semi-positive P-invariant of the SGSPN.*

Proof. Since the incidence functions concerning places of i remain unchanged by synchronization of transitions and the corresponding vector X_i is padded with zeros for all places not contained in i, $X_iC = 0$ is satisfied. Hence, X_i is a semi-positive P-invariant [21]. □

On the other hand, not all semi-positive P-invariants of the SGSPN can be derived from the semi-positive P-invariants of its components, as the example net in Fig. 1 shows. A generating set of P-invariants in A is given by a unit vector; the same holds for B. Their corresponding vectors X_A and X_B are given in the table below. Nevertheless, the SGSPN has additional semi-positive P-invariants z_1, z_2, z_3 which cannot be obtained as a linear combination of X_A and X_B. The additional P-invariants can be seen as global constraints imposed by superposition.

	a_1	a_2	a_3	b_1	b_2	b_3
X_A	1	1	1	0	0	0
X_B	0	0	0	1	1	1
z_1	1	0	0	1	0	0
z_2	0	1	0	0	1	0
z_3	0	0	1	0	0	1

It is well-known that semi-positive P-invariants are useful to calculate upper limits for the number of tokens on places which belong to their support, i.e., $limit(p) = \sum_{p' \in P} x(p')M_0(p')/x(p)$ gives an upper limit of p if $x(p) > 0$. Consequently, we suggest to obtain upper limits in this manner and to obey them during the generation of state spaces for the isolated components. These upper limits can be very effective: for the net in Fig.1, places (b_1, b_2, b_3) are limited to $(p+m, p+m, p)$ for the initial marking $M_0^B = (p+m, p+m, 0)$ due to P-invariants z_1, z_2, z_3. Again, by combinatorial arguments, we have:

$$|\mathcal{PS}| = \binom{3+p-1}{p} \sum_{i=0}^{p} \binom{2+i-1}{i} = \binom{p+2}{p}^2 \geq \binom{p+2}{p} = |\mathcal{TRS}| \quad (3)$$

This reduces $|\mathcal{PS}|$ significantly compared to (1), but it is still much larger than $\mathcal{TRS}$. Apart from reducing the size of component state spaces for efficiency reasons, such upper limits can be necessary to ensure finiteness of component state spaces. Christensen et al. [10] observe that a SGSPN with finite $\mathcal{TRS}$ can have components with infinite state spaces. In this case, only a finite subset of the infinite component state space is in fact relevant in $\mathcal{TRS}$. If the unbounded places in the isolated components are covered by P-invariants in the SGSPN, this problem is solved by deriving upper limits.

4 A Permutation to distinguish $\mathcal{TRS}$ from $\mathcal{PS} \backslash \mathcal{TRS}$

In this section, we consider an approach which is orthogonal to the upper limits regarded above, in order to obtain an iterative analysis which focuses on reachable states. We accept that a structured representation can contain unreachable states for the price of a memory efficient matrix representation. Instead, we attack the negative consequences of unreachable states for numerical analysis. Unreachable states increase the size of iteration vectors, which is crucial for the applicability of the method in terms of memory requirements. Furthermore, they cause useless multiplications of matrix and vector elements, since unreachable states permanently stay at zero probability. Whenever vector entries corresponding to unreachable states are considered in an iteration step, it is a waste of time. Finally if unreachable states are not distinguished from reachable states, this implies an initial distribution $x^{(0)}$ with $x^{(0)}(M) = 1$ if $M = M_0$ and 0 otherwise, as in [15], because for $M \notin \mathcal{TRS}$ the initial probability should be zero. However, $x^{(0)}$ should be chosen carefully due to its impact on convergence. Separating $\mathcal{TRS}$ and $\mathcal{PS} \backslash \mathcal{TRS}$ gives the freedom of choice for $x^{(0)}$ on $\mathcal{TRS}$. Selection of a "good" initial distribution is surely model dependent, such that the degree of freedom obtained by the knowledge of $\mathcal{TRS}$ is valuable, e.g., approximate analysis methods can be used to derive a sophisticated initial distribution provided it fulfills $\pi^{(0)}(M) = 0$ for $M \notin \mathcal{TRS}$.

The main problems in handling unreachable states during iteration are that their vector positions are mixed with positions of reachable states and $\mathcal{TRS}$ is not known. Since the latter problem is solved by the algorithm given below, let us assume for now the set $\mathcal{TRS}$ is known. In this case, one can reorder states according to their reachability and save space for unreachable states, e.g., for the net in Fig. 1 with $p = 1$ and $m = 0$, we obtain component state spaces:

$\mathcal{TRS}^A$				$\mathcal{TRS}^B$			
a_1	a_2	a_3	$index$	b_1	b_2	b_3	$index$
0	0	1	0	1	1	0	0
0	1	0	1	0	0	2	1
1	0	0	2	0	1	1	2
				1	0	1	3
				0	2	0	4
				2	0	0	5

$|\mathcal{PS}| = 18$, but it is easy to verify that $\mathcal{TRS}$ contains just 3 states.

$\mathcal{TRS}$						$index$	
a_1	a_2	a_3	b_1	b_2	b_3	$mix(M^A, M^B)$	$= M^A \cdot 6 + M^B$
0	0	1	1	1	0	$mix(0,0)$	$= 0$
0	1	0	0	1	1	$mix(1,2)$	$= 8$
1	0	0	1	0	1	$mix(2,3)$	$= 15$

This means that an iterative method based on Theorem 1 uses iteration vectors of dimension 18, where only positions 0, 8, and 15 are effectively used. Clearly, one can store the values of these vector positions in a vector of length $|\mathcal{TRS}| = 3$ instead, provided one performs an appropriate address transformation. This is formally described by defining a permutation which reorders states according to their reachability.

Definition 5 *For a SGSPN with $|\mathcal{PS}| = k$, a bijective function $perm : \{0, 1, \ldots, k-1\} \longrightarrow \{0, 1, \ldots, k-1\}$ is a $\mathcal{TRS}$-permutation if*

- $\forall M_x \in \mathcal{PS} : perm(x) < |\mathcal{TRS}| \iff M_x \in \mathcal{TRS}$
- $\forall M_x, M_y \in \mathcal{TRS} : perm(x) < perm(y) \iff x < y$

Note that several $\mathcal{TRS}$-permutations exist for the $\mathcal{PS}$ of a given SGSPN, since the definition requires only reachable states to be mapped in an order preserving way into the set $\{0, 1, \ldots, |\mathcal{TRS}| - 1\}$, the mapping of unreachable states into the set $\{|\mathcal{TRS}|, \ldots, k-1\}$ is bijective but not necessarily order preserving. Such a $\mathcal{TRS}$-permutation can be described by a $(k \times k)$ matrix $\mathcal{P}$ with

$$\mathcal{P}(i,j) = \begin{cases} 1 & \text{if } j = perm(i) \\ 0 & \text{otherwise} \end{cases}$$

Let $x_p = x\mathcal{P}$ denote the permuted vector of an iteration vector x, then the following transformation is applied to an arbitrary iteration method with iteration matrix H:

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

$perm$ is bijective, so $\mathcal{P}^{-1}$ exists, and for permutation matrices also holds $\mathcal{P}^{-1} = \mathcal{P}^T$. In Jacobi iteration $H_J = \bar{Q} D^{-1}$ for $Q = \bar{Q} - D$, where $\bar{Q}$ is the transition matrix and D the matrix with diagonal values of Q.

$$\begin{aligned} & x_p^{(n+1)} = x_p^{(n)} \mathcal{P}^T (\bar{Q} D^{-1}) \mathcal{P} \\ \Longleftrightarrow \quad & x_p^{(n+1)} = x_p^{(n)} \mathcal{P}^T \bar{Q} (\mathcal{P}\ \mathcal{P}^T) D^{-1} \mathcal{P} \\ \Longleftrightarrow \quad & x_p^{(n+1)} = x_p^{(n)} (\mathcal{P}^T \bar{Q} \mathcal{P})\ D_p^{-1} \end{aligned}$$

where $D_p^{-1} = \mathcal{P}^T D^{-1} \mathcal{P}$ is again a diagonal matrix and diagonal values are permuted according to $perm$. The practical implications of employing a $\mathcal{TRS}$-permutation are that iteration vectors $x_p^{(n)}, x_p^{(n+1)}$ can be represented by arrays of size $|\mathcal{TRS}|$, where $perm(x) = i$ gives the appropriate position i for a state $M_x \in \mathcal{TRS}$, as the probability of unreachable states is known to be zero. Furthermore, D_p^{-1} can be represented by an array of size $|\mathcal{TRS}|$ in a similar way, and the same holds for $\mathcal{P}, \mathcal{P}^T$. In fact, we show at the end of this section that it is sufficient to use a single integer array of size $|\mathcal{TRS}|$ to represent both, $\mathcal{P}$ and $\mathcal{P}^T$, for numerical analysis.

In this way, no component of the modified iteration scheme is of size $|\mathcal{PS}|$ any more, and we cured the problem of oversized iteration vectors. The second negative implication of $|\mathcal{PS}| >> |\mathcal{TRS}|$, namely the useless computations for unreachable states during iteration, can easily be avoided, if computations are performed only for states in the first part of $x_p^{(n)}$, where reachable states are located.

Of course, if $|\mathcal{TRS}|$ is almost of size $|\mathcal{PS}|$, the suggested approach does not pay off, but in this case the algorithm can easily fall back to the standard iteration scheme without permutation, thus avoiding overhead caused by a permutation.

Exploration of $\mathcal{TRS}$ and generation of $\mathcal{P}$ The definition of a $\mathcal{P}$ presupposes that $\mathcal{TRS}$ is known. In this section, we formulate a search algorithm to compute $\mathcal{TRS}$ which profits from the structured representation. Assume a structured representation is given for a certain model with initial state M_0. Since diagonal values D in Q are irrelevant for $\mathcal{TRS}$-exploration, we focus on state transition matrix $\bar{Q}$. A basic building block for $\mathcal{TRS}$-exploration is the computation of successor states. Successor states can be reached due to local or synchronized transitions. Considering local transitions, let $\bar{Q}_l = \bigoplus_{i=0}^{N-1} \bar{Q}_l^i$. Def. 4 ensures that $\bar{Q}_l = \sum_{i=0}^{N-1} I_{l^i} \otimes \bar{Q}_l^i \otimes I_{r^i}$ where $l^i = \prod_{j=i+1}^{N-1} k^j$ and $r^j = \prod_{j=0}^{i} k^j$. Since all entries in $I_{l^i}, \bar{Q}_l^i, I_{r^i}$ are nonnegative, the entries of the resulting matrix $I_{l^i} \otimes \bar{Q}_l^i \otimes I_{r^i}$ are nonnegative, and hence the summation of such terms for $\bar{Q}_l$ behaves like a logical OR[2]. So all state transitions which result from the firing of local transitions $T^i \backslash TS$ in a component i (followed by a possibly empty firing sequence of immediate local transitions in i) are specified

[2]In fact matrices $\bar{Q}_l^i, \bar{Q}_t^i$ can also be mapped to boolean matrices and $+, *$ to $\wedge, \vee$ for $\mathcal{TRS}$ analysis.

in the corresponding term $I_{l^i} \otimes \bar{Q}_l^i \otimes I_{r^i}$. Matrices I_{l^i}, I_{r^i} have nonzero entries only on their diagonal, so considering the definition of tensor product directly yields the following characterization of nonzero entries in $\bar{Q}_l$.

$$\bar{Q}_l(x,y) \neq 0 \Leftrightarrow \exists i \in IS: \ \bar{Q}_l^i(x^i, y^i) \neq 0 \wedge \forall j \in IS \backslash \{i\} : x^j = y^j \tag{4}$$

where $x = mix(x^{N-1}, \ldots, x^0)$ and $y = mix(y^{N-1}, \ldots, y^0)$. For a synchronized transition t with $\bar{Q}_t = \bigotimes_{i=0}^{N-1} \bar{Q}_t^i$ we obtain a similar result, since $\bar{Q}_t^j = I_j \ \forall j \notin IC(t)$.

$$\bar{Q}_t(x,y) \neq 0 \Leftrightarrow \forall i \in IC(t) : \bar{Q}_t^i(x^i, y^i) \neq 0 \wedge \forall j \notin IC(t) : x^j = y^j \tag{5}$$

Again, this follows directly from Def. 4 and the fact that $\prod a_i \neq 0 \iff \forall i : a_i \neq 0$. Since the tensor product is based on a mixed radix number representation, value y of a successor state M_y for a given state M_x can be obtained from x in the following way: in case of local transitions, the value y for a $\bar{Q}_l(x,y) \neq 0$ is given by $y = mix(x^1, \ldots, x^{i-1}, y^i, x^{i+1}, \ldots, x^n)$. In case of a synchronized transition t, $y = mix(z^{N-1}, \ldots, z^0)$ where $z^i = y^i$ if $i \in IC(t)$ and $z^i = x^i$ otherwise. Since $x = mix(x^{N-1}, \ldots, x^0) = \sum_{i=0}^{N-1} x^i \cdot g_i$ and $y = \sum_{i \notin IC(t)} x^i \cdot g_i + \sum_{i \in IC(t)} y^i \cdot g_i$, y can be obtained from x by $y = x + \sum_{i \in IC(t)} (y^i - x^i) \cdot g_i$. Note that $(y^i - x^i) \cdot g_i$ is a local transformation, such that in case of sparse matrix representations of $\bar{Q}_l^i$, resp. $\bar{Q}_t^i$, we can store $\bar{Q}_l^i$ with $\bar{Q}_l^i(x^i, (y^i - x^i) \cdot g_i)$ resp. $\bar{Q}_t^i(x^i, (y^i - x^i) \cdot g_i)$ on the same space instead. This reduces the effort for the calculation of a successor state to $|IC(t)|$ additions, no multiplications are necessary.

The following $\mathcal{TRS}$-exploration algorithm integrates the computation of successor states based on the structured representation into the standard search algorithm for state space exploration by traversing the reachability graph, e.g., [6]:

Input: Matrices of a structured representation
Program:
Init: $\mathcal{TRS} = \{M_0\}$, $S = \{M_0\}$
begin
  while not empty S
    take M_x out of S
    decode M_x into $(M_x^{N-1}, \ldots, M_x^0)$ by mix^{-1}
    foreach component $i \in IS$
      foreach $\bar{Q}_l^i(M_x^i, M_y^i) \neq 0$
        $M_y = M_x + (M_y^i - M_x^i) \cdot g_i$
        if $M_y \notin \mathcal{TRS}$
        then insert M_y in $\mathcal{TRS}$ and S
    foreach $t \in TS$
      if $\forall i \in IC(t) : \exists \bar{Q}_t^i(M_x^i, M_y^i) \neq 0$
(*)   then foreach combination of elements $\bar{Q}_t^i(M_x^i, M_y^i) \neq 0$ over $IC(t)$
        $M_y = M_x + \sum_{i \in IC(t)} (M_y^i - M_x^i) \cdot g_i$
        if $M_y \notin \mathcal{TRS}$
        then insert M_y in $\mathcal{TRS}$ and S
end

If firing of a synchronized transition t at M_x yields one tangible state, every row $\bar{Q}_t^i(M_x^i, .)$ of a component in $IC(t)$ contains exactly one nonzero entry and hence only one combination of these elements exists in line ($*$). If $M_x[t>$ enables several sequences of immediate transitions, reaching a set of tangible markings $Z = \{M_1, M_2, \ldots, M_a\}$, then there are rows $\bar{Q}_t^i(M_x^i, .)$ which provide several nonzero entries, and in line ($*$) the algorithm computes any combination where exactly one entry is chosen from each row $\bar{Q}_t^i(M_x^i, .)$ over $i \in IC(t)$. This is necessary to reach all markings in Z from M_x.

Correctness of the algorithm is simple to see, since the underlying search algorithm of [6] is correct and the calculation of successor successor states follows Eqs (4) and (5), as discussed above. Efficiency of the algorithm relies on the choice of appropriate data structures for $\mathcal{TRS}$ and S. Space efficiency requires to minimize space for $\mathcal{TRS}$ and S in order to keep the algorithm applicable for large state spaces. Time efficiency requires fast insert and member operations on $\mathcal{TRS}$, and fast insert and delete operations on S, since these operations, especially the member operation on $\mathcal{TRS}$, are applied extremely often. The data structures must compromise between these diverting goals. In GSPN analysis the size of $\mathcal{TRS}$ is generally unknown. This is different in the context of structured representations, since $\mathcal{TRS} \subseteq \mathcal{PS}$. mix is a bijective function, which gives a perfect hash function; no collisions can occur. In consequence we suggest hashing as far as $\mathcal{TRS}$ is concerned. A bit-vector of length $\mathcal{PS}$ with 1 bit per state is sufficient, because only the information whether M_x is or is not element of $\mathcal{TRS}$ has to be stored at position x. Hence, member and insert operations for $\mathcal{TRS}$ are in $O(1)$. The dimension of the hash table is $|\mathcal{PS}|$, which can be critical in terms of memory requirements, but reasonably large state spaces can be explored, because a single entry requires only 1 bit of memory. For space considerations, we assume 8 bytes for a double-precision value and 4 bytes for an integer value; these values are realistic on workstations. We can relate the space for a bit-vector to the space for an iteration vector of dimension $|\mathcal{TRS}|$ used in the numerical analysis. If $\frac{|\mathcal{TRS}|}{|\mathcal{PS}|} \geq \frac{1}{64} = 0.015625$, i.e., if at least 1.5 % of $\mathcal{PS}$ is reachable, then the hash table requires less(!) memory than one iteration vector. So a bit-vector is presumedly uncritical for most applications.

Due to function mix, any state M_x is coded into a single integer value x, such that the set S can be represented by a stack which contains just integer values. This is efficient in terms of memory compared to storing vectors of component states. S requires space for at most $|\mathcal{TRS}|$ integer values, which is at most 0.5 of the amount used for an iteration vector. Push and pop operations on S are in $O(1)$.

In order to calculate the time complexity of this algorithm, we consider the effort per state and per arc in the reachability graph separately. For each state in $\mathcal{TRS}$, a decoding into component states is performed which takes N division and modulo operations. Enabling tests require to consider N matrices for local transitions and $\sum_{t \in TS} |IC(t)| \leq |TS|N$ matrices for synchronized transitions. In summary $O(N|TS| + N)$ operations are performed per state. Per arc, at most 1 member and insert operation on $\mathcal{TRS}$ and 1 push operation on S are performed together with 1 addition in case of a local transition or $|IC(t)| \leq N$ additions in case of a synchronized transition t. So at most $O(N)$ operations are necessary per arc. Since every state $M \in \mathcal{TRS}$ is inserted and removed from S exactly once, such that all of its outgoing arcs are considered exactly once, we obtain, in summary, a time complexity of $O(N|TS||\mathcal{TRS}| + N|E|)$ where E denotes the number of arcs in the TRG. If the SGSPN does not contain redundant transitions, i.e., if there exists no $M, M' \in \mathcal{TRS}$ with multiple arcs from M to M', then $|E|$ is limited by the number of nonzero entries in the $(\mathcal{TRS} \times \mathcal{TRS})$ generator matrix of the associated CTMC. Although, matrices are usually extremely sparse, such that $|E| << |\mathcal{TRS}|^2$ holds, we exploit $|E| \leq |\mathcal{TRS}|^2$ to estimate the worst case complexity as $O(N|\mathcal{TRS}|^2 + N|TS||\mathcal{TRS}|)$. Memory requirements for $\mathcal{TRS}$ and S remain less than the space for 1.5 iteration vectors provided at least 1.5% of $\mathcal{PS}$ are reachable.

In order to obtain $\mathcal{P}$, the hash table is transformed into an integer vector of length $|\mathcal{TRS}|$ which contains the indices of all reachable states in increasing order. A single vector is sufficient to represent $\mathcal{P}$ and $\mathcal{P}^T$, since the entry x at vector position i gives $perm^{-1}(i)$, and for $perm(x)$ a binary search with logarithmic time complexity yields position i if $x \in \mathcal{TRS}$ or denotes that $x \notin \mathcal{TRS}$, where the latter tells an iteration method that x is not reachable and thus irrelevant. For the example discussed above, we get $perm(0) = 0, perm(8) = 1$, and $perm(15) = 2$, so $\mathcal{P}$ can be represented by the vector $(0, 8, 15)$. Note that once a vector for $\mathcal{P}$ is computed, the hash table is not used anymore and its space is freed.

5 A numerical analysis method

In this section, we compose the results of the previous section to a new numerical analysis method, which computes the steady state distribution of the CTMC underlying a SGSPN. SGSPNs are restricted in that the tangible reachability sets of their components have to be finite for the structured representation. In the context of numerical analysis, all $\mathcal{TRS}^i$ have to be strongly connected, which is a necessary condition for ergodicity of the associated CTMC [15]. The algorithm is new in that it uses a memory efficient representation of the iteration matrix and(!) a memory efficient representation of the iteration vector, which, in combination, allows to analyze SGSPN models with tangible reachability sets of several million states.

Input: SGSPN
Output: steady state distribution if convergence is obtained
1. Calculate upper limits for the number of tokens on places in P.
2. Generate $\mathcal{TRS}^i$ and matrices $\bar{Q}^i_l$, $\bar{Q}^i_t$ for all components i, which includes elimination of vanishing markings. Generation of $\mathcal{TRS}^i$ obeys upper limits of step 1.
3. Explore the $\mathcal{TRS}$ of the SGSPN on state transition matrix $\bar{Q}$ of $\mathcal{PS}$ and generate permutation matrix $\mathcal{P}$ from $\mathcal{TRS}$ as described in Sec. 4, last paragraph.
4. Choose initial distribution on $\mathcal{TRS}$, e.g., uniform distribution.
5. If $|\mathcal{TRS}| << |\mathcal{PS}|$ perform an iterative method employing the permutation matrix $\mathcal{P}$.
In case of Jacobi or JOR generate D_p^{-1} from $\bar{Q}$ for all elements of $\mathcal{TRS}$, for power method generate D_p respectively.
Jacobi

$$x_p^{(n+1)} = \left[x_p^{(n)} \mathcal{P}^T (\bigoplus_{i=0}^{N-1} \bar{Q}^i_l) \mathcal{P} + \sum_{t \in TS} x_p^{(n)} \mathcal{P}^T (W(t) \bigotimes_{i=0}^{N-1} \bar{Q}^i_t) \mathcal{P} \right] D_p^{-1}$$

Jacobi overrelaxation (JOR) choose relaxation factor $\omega \in]0, 2[$

$$x_p^{(n+1)} = (1-\omega) x_p^{(n)} + \omega \left[x_p^{(n)} \mathcal{P}^T (\bigoplus_{i=0}^{N-1} \bar{Q}^i_l) \mathcal{P} + \sum_{t \in TS} x_p^{(n)} \mathcal{P}^T (W(t) \bigotimes_{i=0}^{N-1} \bar{Q}^i_t) \mathcal{P} \right] D_p^{-1}$$

Power method let $\delta = 0.99/max_j|D_p(j,j)|$ and $D' = \mathcal{P}^T(I - \delta D_p)\mathcal{P}$

$$x_p^{(n+1)} = \delta \left[x_p^{(n)} \mathcal{P}^T (\bigoplus_{i=0}^{N-1} \bar{Q}_l^i) \mathcal{P} + \sum_{t \in TS} x_p^{(n)} \mathcal{P}^T (W(t) \bigotimes_{i=0}^{N-1} \bar{Q}_t^i) \mathcal{P} \right] + x_p^{(n)} D'$$

with normalization of $x_p^{(n+1)}$ until convergence is observed.
6. Otherwise perform an ordinary iterative method.
In case of Jacobi or JOR generate D^{-1} from $\bar{Q}$ for all elements of $\mathcal{PS}$, for power method generate D respectively.
Jacobi

$$x^{(n+1)} = \left[x^{(n)} \bigoplus_{i=0}^{N-1} \bar{Q}_l^i + \sum_{t \in TS} x^{(n)} (W(t) \bigotimes_{i=0}^{N-1} \bar{Q}_t^i) \right] D^{-1}$$

Jacobi overrelaxation (JOR) choose relaxation factor $\omega \in]0,2[$

$$x^{(n+1)} = (1-\omega) x^{(n)} + \omega \left[x^{(n)} \bigoplus_{i=0}^{N-1} \bar{Q}_l^i + \sum_{t \in TS} x^{(n)} (W(t) \bigotimes_{i=0}^{N-1} \bar{Q}_t^i) \right] D^{-1}$$

Power method let $\delta = 0.99/max_j|D(j,j)|$ and $D' = \mathcal{P}^T(I - \delta D)\mathcal{P}$

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

with normalization of $x_p^{(n+1)}$ until convergence is observed.

For step 1, upper limits can be derived from a generating set of semi-positive P-invariants obtained by the algorithm in [20], or alternatively by solving $|P|$ (integer) linear programming problems: for each place $p \in P$ maximize $M(p)$ under conditions $M = M_0 + C\sigma$ where $\sigma \geq 0$, and $M \geq 0$. A trivial initial solution is given by $M = M_0$ and $\sigma = 0$. Although we are interested in an integer solution, we might trade the quality of the computed limit for the computation time and use linear programming, for which a solution is possible in polynomial time in theory [18]. However for practical applications, the simplex method is usually employed.

The generation of $\mathcal{TRS}^i$ for an isolated component i in step 2 follows the conventional algorithms for state space exploration, elimination of vanishing markings and matrix generation, but it additionally obeys upper limits imposed by P-invariants of the SGSPN. These limits can be necessary to ensure finiteness of component state space as the example in Sec. 6 shows and can help to reduce the size of $\mathcal{TRS}^i$.

For step 3, the exploration of $\mathcal{TRS}$ based on $\bar{Q}$ is described in Sec. 4. The resulting representation of $\mathcal{P}$ and $\mathcal{P}^T$ is a single integer-vector of length $|\mathcal{TRS}|$. The $\mathcal{P}$ vector contains all states of $\mathcal{TRS}$ in increasing order, it is a $\mathcal{TRS}$-permutation (cf. Sec. 4). Note that the bit-vector employed for $\mathcal{TRS}$-exploration is not used in subsequent steps, so its space is freed. Only the $\mathcal{P}$ vector is necessary.

Diagonal values for vector D_p^{-1} can be calculated as a by-product of $\mathcal{TRS}$-exploration or in a preprocessing step which sums up all nonzero row-entries in $\bar{Q}$, considering all states of $\mathcal{TRS}$. Note that D_p^{-1} is a vector of length $|\mathcal{TRS}|$, and, since its entries are already permuted according to $\mathcal{P}$, the multiplication with the resulting vector of $x_p^{(n)} \mathcal{P}^T \bar{Q} \mathcal{P}$ does not consider $\mathcal{P}$ any more. If the power method is applied, D_p instead of D_p^{-1} has to be generated analogously.

For step 4, the knowledge of $\mathcal{TRS}$ allows to choose an arbitrary initial distribution on $\mathcal{TRS}$ with respect to the applied iteration method, since some iteration methods are sensitive to zero initial probabilities for states in $\mathcal{TRS}$ [24]. The structured representation of $\bar{Q}$ and the knowledge of diagonal values allows to perform the power method as well as Jacobi iteration or Jacobi overrelaxation (JOR).

Furthermore, different implementations are possible to perform the basic vector-matrix multiplication if the matrix is represented by a tensor sum or product. For step 6, we suggest to follow the method used in [22, 23, 24], which enumerates the nonzero matrix entries in a specific order, following the tensor operation. A tensor product for synchronized transitions is performed as $\bigotimes_{i=0}^{N-1} \bar{Q}_t^i = \prod_{i=0}^{N-1} I_{l^i} \otimes \bar{Q}_t^i \otimes I_{r^i}$ by computing a sequence of more simple multiplications: $y_0 = W(t)x^{(n)}$, $y_{i+1} = y_i I_{l^i} \otimes \bar{Q}_t^i \otimes I_{r^i}$ for $0 \leq i < N$. The basic idea for these multiplications is to consider each nonzero entry $\bar{Q}_t^i(M_x^i, M_y^i)$ and to enumerate all states of the other components to obtain the corresponding matrix entries in $I_{l^i} \otimes \bar{Q}_t^i \otimes I_{r^i}$. This enumeration results in some simple index transformations due to the underlying mixed radix number representation. A tensor sum is handled analogously according to $\bigoplus_{i=0}^{N-1} \bar{Q}_l^i = \sum_{i=0}^{N-1} I_{l^i} \otimes \bar{Q}_l^i \otimes I_{r^i}$. This method is efficient if $|\mathcal{PS}| \approx |\mathcal{TRS}|$. In this case, it is advisable to use iteration vectors of size $|\mathcal{PS}|$ and to perform an iteration method without permutation. The initial distribution chosen on $\mathcal{TRS}$ in step 4 is then projected on the corresponding subset of $\mathcal{PS}$. A single multiplication of a vector with a tensor product requires $|\mathcal{PS}| \sum_{i=0}^{N-1} k^i$ multiplications [24], assuming dense matrices, such that the complexity of an iteration step is in $O((N + |TS|)(|\mathcal{PS}| \sum_{i=0}^{N-1} k^i))$ for step 6.

If $|\mathcal{PS}| >> |\mathcal{TRS}|$, which frequently happens due to synchronization, efficiency requires that an algorithm considers just nonzero matrix entries $\bar{Q}(x, y)$ where $x, y \in \mathcal{TRS}$. The algorithm of [22, 23, 24] is inefficient if applied in combination with a $\mathcal{TRS}$-permutation, because intermediate vectors y_i necessary for synchronized transitions are of dimension $|\mathcal{PS}|$; their nonzero entries are not restricted to $\mathcal{TRS}$. The algorithm does not focus on the relevant $(\mathcal{TRS} \times \mathcal{TRS})$ submatrix contained in the structured representation of $\bar{Q}$. Hence, the algorithm in step 5 performs the vector matrix multiplications in $x_p^{(n)}\mathcal{P}^T(\bigoplus \bar{Q}_l^i)\mathcal{P}$ and $x_p^{(n)}\mathcal{P}^T(W(t) \bigotimes_{i=0}^{N-1} \bar{Q}_t^i)\mathcal{P}$ by enumerating all elements M in $\mathcal{P}$ ($= \mathcal{TRS}$) and performing multiplications only for the corresponding rows in $\bigoplus_{i=0}^{N-1} \bar{Q}_l^i$ and $\bigotimes_{i=0}^{N-1} \bar{Q}_t^i$. In particular for each row $M \in \mathcal{TRS}$, matrix entries $\bar{Q}(M, M') \neq 0$ are explicitly generated and multiplied with $x_p^{(n)}(M)$. In order to add the resulting value on the iteration vector $x_p^{(n+1)}$ at the permuted position of M', $i = perm(M')$ must be computed. Since states in $\mathcal{P}$ are ordered, we can employ binary search on $\mathcal{P}$ to compute $i = perm(M')$ in $\mathrm{O}(\log(|\mathcal{TRS}|))$. Since $\bar{Q}$ is a state transition matrix, for any nonzero $\bar{Q}(M, M')$ holds that $M \in \mathcal{TRS} \Rightarrow M' \in \mathcal{TRS}$. In this way the search effort to calculate $i = perm(M')$ can be restricted to reachable states. Obviously, $\mathcal{P}$ provides $x = perm^{-1}(i)$ in $O(1)$ by accessing $\mathcal{P}$ at position i. Note that the search effort is the price for a memory efficient representation of $\mathcal{P}$ and $\mathcal{P}^T$, and that memory requirements are the main bottleneck in the numerical analysis of large CTMCs.

This leads to a time complexity of $O(|\mathcal{TRS}|^2(N + log(|\mathcal{TRS}|)) + |\mathcal{TRS}|N|TS|)$ per iteration in step 5, because for every state in $\mathcal{TRS}$ at most $N|TS|$ matrices are checked for enabled tests and at most $|\mathcal{TRS}|^2$ nonzero matrix entries[3] require

[3]Under the assumption that each nonzero matrix entry results from firing of just one timed

at most N additions for coding plus $log(|\mathcal{TRS}|)$ for $i = perm(M')$. Note that the iteration in step 5 mainly depends on $\mathcal{TRS}$ and the iteration in step 6 depends on $\mathcal{PS}$. So clearly, there is a tradeoff between both approaches. The complexity of the $\mathcal{TRS}$-exploration in step 3 is less than a single iteration step of step 5, since the search effort for the permutation is missing.

6 Analysis of an example SGSPN

In this section, we consider a GSPN model of a parallel communication software system given by Woodside and Li [26]. Figure 2 shows the complete GSPN model which models the one-way transfer of messages, originating at the sender side and being conveyed at the receiver side. Timed transitions are denoted by boxes, immediate transitions by black bars. Indices of places and transitions are lifted to enhance readability, e.g., $p_{32} = p32$. Following [26], messages are processed by two tasks, implementing the ISO session and transport layers at each end, and messages are conveyed between user and layer tasks by "courier" tasks which are active buffers for a single data unit. The transport level fragments data, which is modeled by having two paths from place $p12$ to $p32$. The path via $t8$ creates and carries fragments to place $p32$, where acknowledgements are generated, but no data is delivered to higher layers. The number of fragments carried via $t8$ is decided in a non-deterministic way by the conflict between transitions $t8$ and $t9$. Once $t9$ has fired, $p12$ is empty and $t8$ is disabled until the sender creates a new message. The path via $t9$ models the transport of the last fragment of each message to $p31$, where it generates an acknowledgement (a token on $p32$) and causes a data token to be delivered upwards via $t27, t28$. The transport window size N is represented by the number of initial tokens at $p14$. The transport buffer space M is specified by $M_0(p13)$, we consider the case $M = 1$ here. Shaded circles denote multiple representations of certain places in order to preserve readability in Fig. 2, i.e., the two shaded circles in the lower left describe a single place $p11$ and the two shaded circles on the lower right describe a single place $p33$.

The model splits quite naturally in 4 components A, B, C, and D, as denoted in Fig. 2. Component A contains the user source and the session layer, including courier tasks on the sender side. Transition $t6$ synchronizes the courier task between the session layer and the transport layer of the sender side, which is part of component B. B includes the transport layer at the sender side, while component C contains the corresponding layer at the receiver side. B and C are synchronized via transitions $t18$, $t19$, and $t20$, which describe the network delay. Finally component D covers the user sink and the session layer, including courier tasks on the receiver side. Note that the model does not show symmetries between components, since only a one-way data transfer is considered. Places px and py are not contained in the original model [26]; these places are structurally implicit and will be additionally introduced later on.

Since we focus on the performance of the algorithm given in Sec. 5 rather than performance indices of this model, information on timing parameters is omitted, cf. [26]. Note that this example cannot be analyzed as it is by the approach given in [15], since it breaks the restriction "firing of synchronized transitions leads to

transition and that a full matrix is given for the worst case.

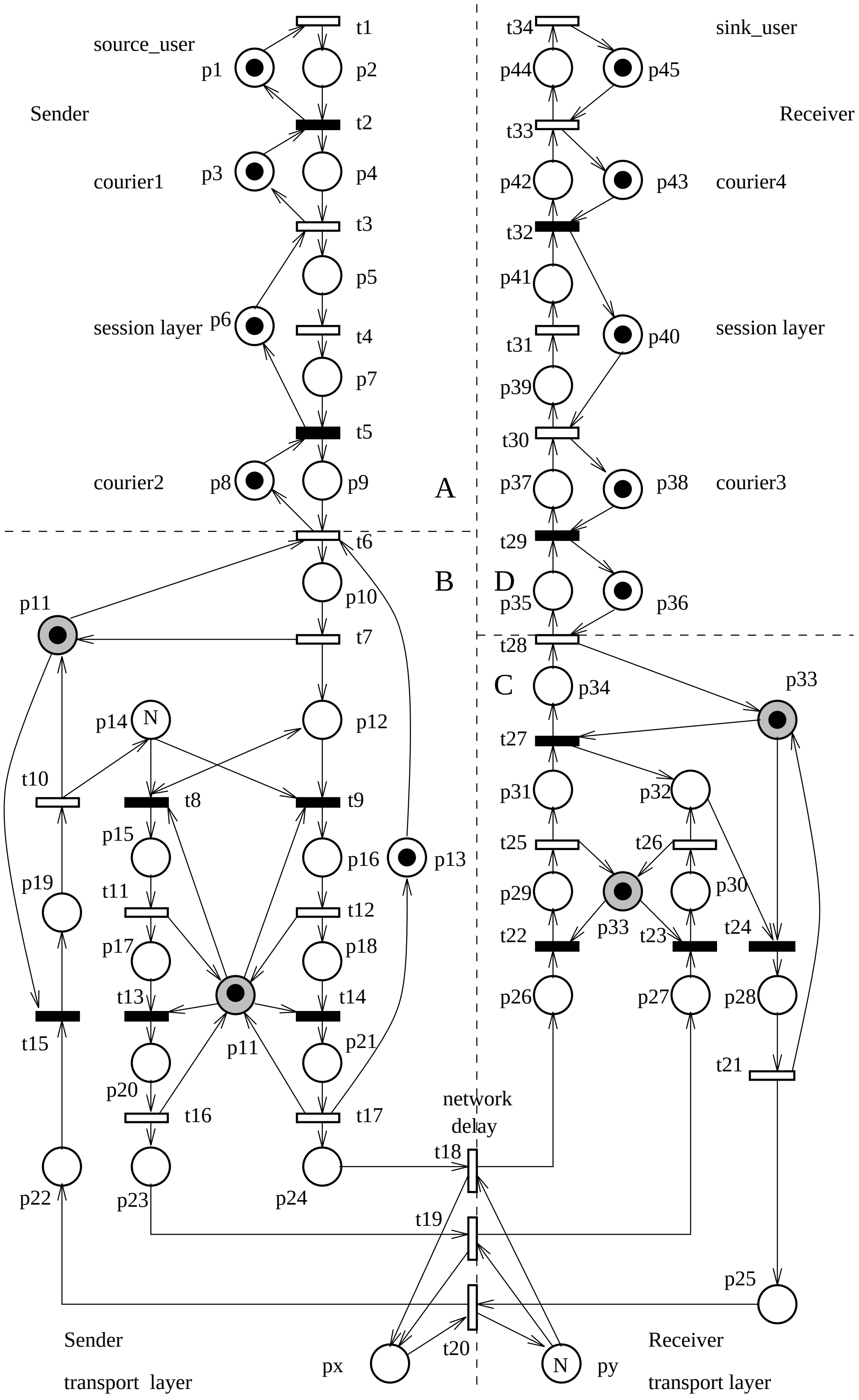

Figure 2: SGSPN model of courier protocol software

tangible states". In principle, the SGSPN can be transformed into a net which fulfills the restriction [9], but this transformation enlarges the number of synchronized transitions drastically, due to the intricate blocking effects implied by immediate transitions $t15$, $t22$, and $t23$. According to the structured representation given in Theorem 1, which is similar to the one given in [15], performance of any algorithm based on it decreases if $|TS|$ is enlarged. Due to Theorem 1, this restriction does not apply for our approach, such that the example can be directly analyzed. Originally places px and py are not contained in the model, such that for the original model upper limits are obtained from P-invariants to achieve finiteness of the isolated component state spaces $\mathcal{TRS}^B$ and $\mathcal{TRS}^C$. The complete SGSPN is covered by P-invariants and contains one P-invariant X which is linearly independent of the P-invariants found in the isolated components namely

$$X(p) = \begin{cases} 1 \ if \ p \in \{p14, p15, \ldots, p32\} \\ 0 \ otherwise \end{cases}$$

P-invariant X allows to impose upper limits/capacities for the number of tokens on places $p14, \ldots, p32$ in components B and C. This is sufficient to ensure finiteness of $\mathcal{TRS}^B$ and $\mathcal{TRS}^C$. However, X can be further exploited: observe that all useful states $M \in \mathcal{TRS}^B$ fulfill

$$\sum_{i=14}^{24} X(p_i)M(p_i) \leq N,$$

resp. for $M \in \mathcal{TRS}^C$

$$\sum_{i=25}^{32} X(p_i)M(p_i) \leq N,$$

and $N = \sum_{i=1}^{46} X(p_i)M_0(p_i)$. By introducing artificial "variables" $M(px), M(py) \in \mathbb{N}_0$ on these inequalities, we obtain

$$1M(px) + \sum_{i=14}^{24} X(p_i)M(p_i) = N,$$

$$1M(py) + \sum_{i=25}^{32} X(p_i)M(p_i) = N,$$

which imposes additional places px and py on net level. Formally, we perform a net transformation by adding places px and py where $M_0(px) = \sum_{i=25}^{32} X(p_i)M_0(p_i)$, $M_0(py) = \sum_{i=14}^{24} X(p_i)M_0(p_i)$ and new arcs are added, such that all synchronized transitions perform the equivalent transformation on $M(px)$ and $\sum_{i=25}^{32} X(p_i)M(p_i)$, resp. $M(py)$ and $\sum_{i=14}^{24} X(p_i)M(p_i)$. The introduction of these additional places avoids a tremendous amount of useless states during generation of $\mathcal{TRS}^B$ and $\mathcal{TRS}^C$, e.g., for $N = 1$, $|\mathcal{TRS}^B| = 16$ and $|\mathcal{TRS}^C| = 6$ if places px and py are given and $|\mathcal{TRS}^B| = 236$ and $|\mathcal{TRS}^C| = 90$ if not.

Table 1 shows the sizes of $\mathcal{TRS}^A$, $\mathcal{TRS}^B$, $\mathcal{TRS}^C$ and $\mathcal{TRS}^D$ in columns A, B, C, and D for increasing values of N. Furthermore, columns $\mathcal{PS}$, $\mathcal{TRS}$ give the cardinalities of these sets and column "ratio" gives the ratio $\mathcal{TRS}/\mathcal{PS}$. Column $NZ(Q)$ gives the number of nonzero entries in $\bar{Q}$, which refer to reachable states and which must be considered in each iteration step. Note that these elements would be stored for a conventional iteration method, but this is not the case for a structured representation. Column $\sum NZ(Q^i)$ gives the number of nonzero entries in the structured

N	A	B	C	D	$\mathcal{PS}$	$\mathcal{TRS}$	ratio	$NZ(Q)$	$\sum NZ(Q^i)$
1	15	16	6	30	43200	11700	0.271	48330	122
2	15	69	27	30	838350	84600	0.101	410160	281
3	15	217	88	30	8593200	419400	0.049	2281620	845
4	15	546	229	30	56265300	1632600	0.030	9732330	2333
5	15	1188	510	30	272646000	5358600	0.020	34424280	5597

Table 1: State spaces of example system for increasing values of N

	CPU time			elapsed time		
N	comp	$\mathcal{TRS}$	iter	comp	$\mathcal{TRS}$	iter
1	0.03	0.67	0.78	1	1	1
2	0.08	3.67	7.13	1	4	8
3	0.23	18.93	40.49	1	19	42
4	0.60	79.02	179.86	1	80	188
5	1.47	271.17	705.77	2	276	1782

Table 2: Computation times of example system for increasing values of N

representation which need to be stored. Note that the percentage of $\mathcal{PS}$ which is in fact reachable clearly indicates that a numerical method, considering $\mathcal{PS}$ instead of $\mathcal{TRS}$, is hopelessly inefficient under these circumstances; for $N \geq 3$ less than 5% of $\mathcal{PS}$ is reachable. The worst case time complexity of the new approach shows a factor $log(|\mathcal{TRS}|)$ for the calculation of $perm^{-1}(x)$ due to binary search on $\mathcal{P}$. Since this search is performed once for each multiplication of a nonzero matrix entry with a vector element, its implementation suggests itself for optimization. Binary search on $\mathcal{P}$ gains in efficiency, if the search interval is reduced, especially if it is significantly smaller than the worst case assumption $|\mathcal{TRS}|$. For this example, our implementation reduces the maximal length of the search interval by one order of magnitude and the mean length is even two orders of magnitude smaller than $|\mathcal{TRS}|$, e.g., for $N = 5$ we get $|\mathcal{TRS}|$ =5358600, a maximal interval length of 126449 and a mean interval length of 12442. The key idea is to exploit the mixed radix number representation, which gives a block structure on $\mathcal{TRS}$, and the sequential enumeration of rows, corresponding to $\mathcal{TRS}$, to obtain tight bounds for the search interval. Nevertheless, the criteria obtained this way are only effective to handle most nets, but do not reduce the worst case time complexity.

Table 2 gives CPU-time and elapsed time in seconds for the exploration of component state spaces in column "comp", the exploration of the complete $\mathcal{TRS}$ in column "$\mathcal{TRS}$" and the time per iteration step performing JOR in column "iter". These times are obtained on a Sparc 4 with 85 MHz CPU and 64 MB primary memory. Diagonal values are computed only once in a preprocessing step and stored in a vector. The bottleneck is clearly the memory requirements of the iteration vectors in step 5. Note that the $\mathcal{TRS}$ exploration is performed without a significant difference between CPU and elapsed time for $N = 5$. On a Sparc station, a double precision value uses 8 bytes and an integer values 4 bytes, such that two iteration vectors, a vector for diagonal values, and an integer vector for $\mathcal{P}$ results in 28 bytes per element in $\mathcal{TRS}$ for the implemented algorithm. Additional space for the nonzero entries of the structured representation is negligible here, as column $\sum NZ(Q^i)$ in Tab. 1 shows. Consequently models up to 2.2 million states can be analyzed on a 64 MB machine without relying on virtual memory significantly. In fact, the results for $N = 4$ (1.6 million states) show no significant difference between CPU-time and elapsed time. For $N = 5$ the algorithm uses about 150 MB and has to rely on secondary memory, which results in a significant amount of time for paging operations. This effect can be seen by the difference between elapsed and CPU-time. The sharp increase of computation times (CPU as well as elapsed time) needs to be related to the increase of $\mathcal{TRS}$ for $\mathcal{TRS}$-exploration and to the increase of $NZ(Q)$ for performing an iteration step. Woodside and Li [26] achieve exact results for $N \leq 2$ and approximate results for $N \leq 7$, but do not specify their machine configuration. The new algorithm allows to compute exact solutions for $N \leq 4$ on a machine with 64 MB primary memory. A solution for $N = 5$ is feasible as well, but the computation relies on sufficient secondary memory, which extends the solution time by a factor of 2.5.

7 Conclusions

In this paper, we consider an analysis technique for SGSPNs. SGSPNs are GSPNs for which the modeler gives a partition into components. Since the analysis technique relies on this partition, it need not be profitable for arbitrary SGSPNs or arbitrary partitions. The problem of finding a suitable partition into components for a given GSPN is not considered here and subject to ongoing research. The example given in Sec. 6 indicates that P-invariants are helpful to improve a partition, however, their availability and relevance for the quality of a partition is clearly model-dependent.

We focus on a numerical analysis technique for CTMCs derived from SGSPNs. The technique is based on a structured description of the generator matrix Q, which describes Q by a sum of tensor products. Structured descriptions based on tensor operations for Q matrices have been developed and successfully employed for various modeling formalisms including SGSPNs as well [5, 15, 22, 23]. The structured description, we propose, is similar to the one in [15], but less restrictive in that it only requires that synchronized transitions have to be timed and the tangible reachability graphs of isolated components (subnets) have to be finite. Our description consists of N+TS tensor products, one for each component and for each synchronized transition. We decided to use a direct representation of diagonal values as a vector in the size of the tangible reachability set, which allows us to use other iteration methods than the power method, e.g., the Jacobi and Jacobi overrelaxation methods.

The main advantage of a structured description of Q is that it is very memory efficient: only a set of relatively small matrices need to be stored. A structured representation of Q provides matrix entries for a set of states, $\mathcal{PS}$, which results from the cross product of its component state spaces. However, the tangible reachability set $\mathcal{TRS}$ of a SGSPN is often only a small subset of $\mathcal{PS}$, due to the synchronization between components; so often $|\mathcal{PS}| >> |\mathcal{TRS}|$.

This effect reduces efficiency and applicability of a structured approach, if not treated adequately. In this paper we propose two means to solve this problem:

1. Constraints imposed by the SGSPN on its components are suitable to restrict the state spaces of isolated components for the context they are embedded in. In particular, a SGSPN can impose P-invariants, which give effective place capacities for the state space generation of components in isolation. This can reduce the size of $\mathcal{PS}$ and ensure finiteness of component state spaces. However, relying on P-invariants alone is not sufficient in general.

2. Orthogonal to this, $\mathcal{PS}$ can be partitioned into the set of reachable states $\mathcal{TRS}$ and unreachable states by an appropriate permutation, such that an iterative numerical method can focus on $\mathcal{TRS}$ alone.

Both ideas are employed in the analysis algorithm, such that SGSPNs can be analyzed on a standard work station where $\mathcal{TRS}$ contains several millions of states and $\mathcal{PS}$ can be larger than $\mathcal{TRS}$ up to 2 orders of magnitude. Additionally the analysis algorithm allows to choose an initial distribution on $\mathcal{TRS}$, e.g., one derived from an approximate technique, a uniform distribution, or P[initial state]=1.0 and 0.0 for all other states. The iteration can be performed according to Jacobi overrelaxation (JOR), Jacobi method, or Power method. The solutions obtained by the new approach are exact, as far as the iterative numerical solutions of CTMCs are exact.

So far, SGSPNs with constant transition weights have been considered. However, transition weights can also be defined for GSPNs, such that they are marking dependent [1, 8]. If applied to SGSPNs, only certain kinds of marking dependent weights can be included into the structured description as is, e.g., the weights of local transitions may depend on the marking of the component they belong to. Since the modeler has the freedom to choose a partition into components, a non-trivial set of examples with marking-dependent weights can be handled well by the approach presented so far. Plateau et al. [22, 23] introduced generalized tensor products to handle the case of arbitrary state-dependent rates for SANs. This generalization, however, imposes a significant extra effort for practical applications [24]. Hence, future work for SGSPNs goes into the integration of a restricted but sufficient set of marking dependent weight functions into numerical algorithms without increasing the algorithm's time complexity significantly. So far, the algorithm given here has been implemented sequentially within a modified QPN-Tool [4]. Future work will also be dedicated to a parallel implementation and an integration into hierarchical concepts.

References

[1] M. Ajmone Marsan, G. Balbo, and G. Conte. A class of generalized stochastic Petri nets for the performance analysis of multiprocessor systems. *ACM Transactions on Computer Systems*, 2(1), May 1984.

[2] V. Amoia, G. De Micheli, and M. Santomauro. Computer-oriented formulation of transition-rate matrices via Kronecker algebra. *IEEE Trans. Reliability*, R-30(2):123–132, June 1981.

[3] G. Balbo, G. Chiola, G. Franceschinis, and G. Molinar-Roet. On the efficient construction of the tangible reachability graph of generalized stochastic Petri nets. In *Int. Workshop on Petri Nets and Performance Models*, pages 136–145. IEEE Computer Society, 1987.

[4] F. Bause and P. Kemper. QPN-Tool for qualitative and quantitative analysis of queueing Petri nets. In G. Haring and G. Kotsis, editors, *Computer Performance Evaluation, Modelling Techniques and Tools, Proc. 7th int. Conf., Vienna, Austria*, LNCS 794, pages 321–334. Springer, 1994.

[5] P. Buchholz. A hierarchical view of GCSPNs and its impact on qualitative and quantitative analysis. *Journal of Parallel and Distributed Computing*, 15:207–224, 1992.

[6] G. Chiola. Compiling techniques for the analysis of stochastic Petri nets. In *Proc. of the 4th Int. Conf. on Modeling Techniques and Tools*, pages 13–27, 1989.

[7] G. Chiola. GreatSPN 1.5 software architecture. In G. Balbo and G. Serazzi, editors, *Computer Performance Evaluation*, pages 121–136. NorthHolland, 1992.

[8] 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.

[9] G. Chiola, S. Donatelli, and G. Franceschinis. GSPNs versus SPNs: what is the actual role of immediate transitions. In *Proc. 4th Int. Workshop on Petri Nets and Performance Models*, pages 20–31. IEEE Computer Society, 1991.

[10] S. Christensen and L. Petrucci. Modular state space analysis of coloured Petri nets. In *Proc. 16th int. Conf. Application and Theory of Petri Nets*, LNCS 935, pages 201–217. Springer, 1995.

[11] G. Ciardo, J. Muppala, and K. Trivedi. SPNP: Stochastic Petri net package. In *Proc. of the 3rd Int. Workshop on Petri Nets and Performance Models*, pages 142–151. IEEE Computer Society, 1989.

[12] G. Ciardo and K.S. Trivedi. A decomposition approach for stochastic Petri net models. In *Proc. 4th Int. Workshop on Petri Nets and Performance Models*, pages 74–83. IEEE Computer Society, 1991.

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

[14] S. Donatelli. Superposed stochastic automata: a class of stochastic Petri nets with parallel solution and distributed state space. *Performance Evaluation*, 18:21–26, 1993.

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

[16] S. Donatelli, M. Ribaudo, and J. Hillston. A comparison of performance evaluation process algebra and generalized stochastic Petri nets. In *Proc. 6th Int. Workshop Petri Nets and Performance Models*, pages 158–168. IEEE Computer Society Press, 1995.

[17] J. Hillston. Compositional Markovian modelling using a process algebra. In W.J. Stewart, editor, *Computations with Markov Chains: Proc. 2nd Int. Work. Numerical Solution of Markov Chains*, pages 177–196. Kluwer Academic Publishers, 1995.

[18] N. Karmarkar. A new polynomial time algorithm for linear programming. In *Combinatorica*, volume 4, pages 373–395. 1984.

[19] C. Lindemann. DSPNexpress: a software package for the efficient solution deterministic and stochastic Petri nets. *Performance Evaluation*, 22, 1995.

[20] J. Martinez and M. Silva. A simple and fast algorithm to obtain all invariants of a generalized Petri net. In C. Girault and W. Reisig, editors, *Application and Theory of Petri Nets*, Informatik Fachberichte 52, 1982.

[21] T. Murata. Petri nets: properties, analysis and application. *Proc. of the IEEE*, 77:541–580, 1989.

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

[23] B. Plateau and J.M. Fourneau. A methodology for solving Markov models of parallel systems. *Journal of Parallel and Distributed Computing*, 12, 1991.

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

[25] W.J. Stewart, K. Atif, and B. Plateau. The numerical solution of stochastic automata networks. *European Journ. of Oper. Res.*, 86:503–525, 1995.

[26] C.M. Woodside and Y. Li. Performance Petri net analysis of communications protocol software by delay-equivalent aggregation. In *Proc. 4th Int. Workshop on Petri Nets and Performance Models*, pages 64–73. IEEE Computer Society, 1991.