Reachability Analysis Based on Structured Representations

Peter Kemper

Informatik IV, Universität Dortmund
44221 Dortmund, Germany
e-mail:kemper@ls4.informatik.uni-dortmund.de

Abstract. Exploration of the reachability set ($\mathcal{RS}$) is one of the crucial building blocks for various analysis methods ranging from model checking to Markov chain (MC) based performance analysis. In the context of MCs, structured representations of state transition matrices using tensor (Kronecker) algebra have been successfully employed to handle the impact of the state space explosion problem. In this paper such structured representations give rise to a new $\mathcal{RS}$ exploration algorithm for superposed generalized stochastic Petri nets and stochastic automata networks. The algorithm employs bitstate hashing with a perfect hash function, i.e. no collisions can occur. Two variations of this algorithm are discussed. Two examples are exercised to demonstrate the benefits of the new algorithm.

1 Introduction

Reachability analysis is one of the analysis methods applicable to Petri nets of various kinds: ordinary and colored, timed and untimed, hierarchical and non-hierarchical, etc. Furthermore it serves as an underlying method for several analysis techniques, e.g. model checking and Markov chain based analysis of stochastic Petri nets. Besides its usefulness the state space explosion problem has been recognized as its main drawback for long. Many approaches exist to handle the impact of large state spaces, among others there are:

Conventional reachability set ($\mathcal{RS}$) exploration is based on a straightforward search algorithm; G. Chiola [8] describes a sophisticated coding of markings and enabling tests exploiting causal dependencies to optimize this approach. The parallelization of conventional RS exploration for massively parallel machines by considering parallel transition firings and parallel member and insert operations for $\mathcal{RS}$ is discussed in [6, 7].

Reducing $\mathcal{RS}$ by avoiding unnecessary interleaving is the key idea for Valmari's stubborn set method [23, 24], Godefroid's sleep set method [14, 15], and combinations of these [15, 25], where many interesting qualitative properties can be analyzed from the reduced $\mathcal{RS}$.

Approximate methods, where certain parts of $\mathcal{RS}$ possibly remain unconsidered, have been developed. Among others, one very successful approach is the bitstate method by Holzmann of which different variations exist [16, 26, 17] and

which allows to analyze extremely large state spaces for the price of completeness.

A modular approach is given by Christensen et al. [9] who consider a decomposition of a (colored) net into a set of components which interact via synchronized transitions (transition fusion). The main idea there is to compute reachability graphs for each component in parallel as far as independent, component specific transitions allow and then to check the enabling of synchronized transitions within all components. This yields one reachability graph per component plus an additional synchronization graph. The underlying assumption is that for a single component with an enabled synchronized transition its environment, given by the rest of components, is likely to disable this transition; hence the generation of the component reachability graphs is tightly coupled for synchronized transitions. The resulting modular representation of $\mathcal{RS}$ is shown to be useful for proving qualitative properties.

Structured representations of the generator matrix Q have been successfully developed in the context of MC based performance analysis. They are employed for the analysis of various hierarchical net formalisms with asynchronous communication by P. Buchholz [3]. For modeling formalism with synchronous interaction fundamental work refers to B. Plateau and coworkers for stochastic automata networks (SANs) [20, 21]. Her results where transferred to the Petri net world by S. Donatelli, who considers superposed stochastic automata in [12] and superposed generalized stochastic Petri nets (SGSPNs) in [13]. Structured representations are also known for stochastic process algebras [4, 5]. Within these structured representations often not all states are in fact reachable as stated in [10, 13, 18] such that the reachability problem arises again. The problem has been addressed in [19], where a solution for the numerical analysis of SGSPNs is demonstrated but the employed $\mathcal{RS}$ exploration algorithm is only briefly sketched. There the focus point is in the performance analysis based on the associated MC and computation of $\mathcal{RS}$ is used as a prerequisite.

In the following we describe the underlying algorithm of [19] for $\mathcal{RS}$ exploration in detail and for the extended context of SGSPNs and SANs. The algorithm follows a modular approach and exploits that (by definition) these modeling formalisms support a decomposition into components which interact via synchronized transitions. From this point of view the new algorithm is related to the modular approach of Christensen et al. [9]. The main difference is that reachability sets of components are explored in complete isolation assuming that synchronized transitions are not disabled by other components. This is advantageous because these state spaces can be explored in parallel and subsequent exploration of the overall $\mathcal{RS}$ can use bitstate hashing. Compared to the work of Holzmann [16, 26, 17] we also use a bit vector as a hash table but with a perfect(!) hash function instead, such that the resulting $\mathcal{RS}$ is exact. A by-product of this hash function is that a marking/state can be uniquely encoded into a single integer value, which reduces the memory requirements of other involved data structures dramatically. An additional improvement of the new algorithm is suggested, which reduces the impact of interleaving. The key idea is to consider

the firing of transitions in certain orders without changing reachability. Similar ideas are discussed in [1] for the efficient elimination of vanishing markings and in [23] to reduce $\mathcal{RS}$. The main difference is that $\mathcal{RS}$ is not(!) reduced here and that sets of states are considered instead of single states. Note that for subsequent MC based performance analysis the complete $\mathcal{RS}$ is required; a reduced $\mathcal{RS}$ as produced by stubborn or sleep set methods is not sufficient for MC based performance analysis.

The paper is organized as follows: Sec. 2 gives the notational background for structured representations of SGSPNs and SANs; in Sec. 3 the new algorithm is described as a variation of the conventional search method, where data structures profit from a given structured representation. Improvements of this algorithm are presented in Sec. 4. Two examples for SGSPNs are taken from literature [7, 10] and exercised in Sec. 5 to demonstrate applicability and efficiency of the new algorithms. Conclusions and prospects to future work finish the paper.

2 Definitions

In this section we will briefly recall definitions and results for tensor algebra, SANs and SGSPNs, in order to fix the notation for subsequent sections and to clarify the similarities between these modeling formalisms at the level of state transition matrices. The notation for SGSPNs follows mainly [13], we assume that the reader is familiar with GSPNs and their dynamic behavior.

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 $T \cap P = \emptyset$, $\pi : T \to \{0, 1\}$ 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, $W : T \to \mathbb{R}^+$ is a function that assigns a weight to each transition, $M_0 : P \to \mathbb{N}_0$ is the initial marking: a function that assigns a nonnegative integer value to each place.

Let $T_i := \{t \in T | \pi(t) = 1\}$ ($T_e := T \backslash T_i$) denote the set of immediate (timed) transitions. Immediate transitions fire with priority over timed transitions. It is possible to define different levels of priorities for T_i but for simplicity we consider just $\pi : T \to \{0, 1\}$. $M[t>M'$ indicates that the system state/marking changes from M to M' due to firing of $t \in T$, $M[t>$ denotes that $t \in T$ is enabled in M and $M[>$ gives the set of enabled transitions. If marking dependent weights shall be considered W modifies to $W : M \times T \to \mathbb{R}^+$. Based on Def. 1 and the firing rule the reachability set ($\mathcal{RS}$), the reachability graph (RG), the tangible reachability set (TRS) and the tangible reachability graph (TRG) can be defined in the usual manner.

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 associated continuous time Markov chain (CTMC) is given by $Q = \bar{Q} - D$ with a diagonal matrix D, $D(i, j) := \sum_k \bar{Q}(i, k)$ if $i = j$ and 0 otherwise. Matrix $\bar{Q}$ represents the reachability relation and additional timing information. Since in this context we are only interested to compute $\mathcal{RS}$, resp. $\mathcal{TRS}$, we can exploit

$$\bar{Q}(M, M') \neq 0 \iff \exists t \in T_e, \sigma \in T_i^* : M[t\sigma>M',$$

where σ is a maximal firing sequence of immediate transitions, and do not further distinguish between numerical values of nonzero entries in $\bar{Q}$.

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, $TS \subseteq \{t \in T | \pi(t) = 0\}$ is the set of synchronized transitions, that are timed by definition. Moreover Π induces on $T \backslash TS$ a partition of transitions. Such a SGSPN contains N components $(P^i, T^i, \pi^i, I^i, O^i, H^i, W^i, M_0^i)$ for $i \in IS := \{0, 1, \ldots, N-1\}$, where $T^i := {}^\bullet P^i \cup P^i{}^\bullet$ 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.

SGSPNs are naturally amenable for a modular $\mathcal{RS}$ analysis, since partition Π induces components, which are GSPNs themselves. In consequence state spaces of components can be explored independently and in parallel by a conventional $\mathcal{RS}$ exploration. The underlying assumption is that enabling of a synchronized transition depends only on the state of component i during generation of $\mathcal{RS}^i$, resp. $\mathcal{TRS}^i$. In [9] Christensen et al. argue that component state spaces $\mathcal{RS}^i$ can be infinite, while the $\mathcal{RS}$ of the complete model is finite. In general we cannot cure this problem; nevertheless if such a critical component i is not covered by P-invariants in isolation but covered by P-invariants within the SGSPN, these global P-invariants are suitable to achieve finiteness of $\mathcal{RS}^i$ by providing place capacities. P-invariants are similarly exploited in [19] to improve efficiency of a numerical method for SGSPNs by enforcing state spaces of components to obey place capacities deduced from global P-invariants. So in the following we assume that a SGSPN is given for which finite component state spaces $\mathcal{RS}^i$ are calculated via conventional $\mathcal{RS}$ exploration.

For GSPNs typically only so called tangible states are relevant and it is common and efficient practice to eliminate vanishing states during generation of TRG. In case of SGSPNs observe that synchronized transitions are timed by definition, which facilitates to obtain the TRG of a SGSPN from component state spaces TRG^i by eliminating vanishing states locally during generation of TRG^i [19]. Generation of TRG^i for an isolated component i yields a state transition matrix Q^i, which can be transformed into a term $Q^i = Q_l^i + \sum_{t \in TS} w(t) Q_t^i$ where Q_l^i contains all entries due to the firing of (timed) local transitions (possibly multiplied by the probability of subsequently firing a sequence of immediate transitions). In matrices Q_t^i for a synchronized transition $t \in TS$ the row sum of any row j is either 1 if t is enabled in M_j^i or 0 otherwise. Q_t^i contains the conditional probabilities of firing a (possibly empty) sequence of immediate transitions in component i under the condition that t is enabled and fires, i.e. $Q_t^i(j,k)$ gives the probability to reach M_k^i from $M_j^i[t>$ via a firing sequence of immediate transitions in component i.

These component matrices are used in a so-called structured representation to describe Q for the overall SGSPN by the help of tensor algebra [11], which itself is based on a mapping function using mixed radix number representation.

Definition 3. Mapping function *mix*
Let $\mathcal{RS}^i := \{0, 1, \ldots, k^i - 1\}$ be some finite sets with arbitrary but fixed constants k^i for all $i \in \{0, 1, \ldots, N-1\}$ and $k = \prod_{i=0}^{N-1} k^i$. Let $\mathcal{PS} := \times_{i=0}^{N-1} \mathcal{RS}^i$ denote the product space. A mapping $mix : \mathcal{PS} \longrightarrow \{0, 1, \ldots, k-1\}$ 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 $(x^{N-1}, \ldots, x^0) \in \mathcal{PS}$ is the mixed radix number representation of $x = mix(x^{N-1}, \ldots, x^0)$ with respect to basis $(k^{N-1}, \ldots, k^0)$. Like any number representation *mix* is bijective and its inverse mix^{-1} can be calculated from $x^i \equiv (mix(x^{N-1}, \ldots, x^1, x^0)/g_i) mod\ k^i$ for each i. In the following, set $\mathcal{RS}^i$ of Def. 3 coincides with the set of reachable states in component i, such that *mix* induces a numbering on $\mathcal{PS}$ as well. Since such a 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.

For the definition of tensor product we follow the notation in [11] but regard only the restricted case of square matrices to keep a concise notation, because only square matrices occur in the context of our modeling formalism.

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 $\bigoplus_{i=0}^{N-1} A^i := \sum_{i=0}^{N-1} I_{l^i} \bigotimes A^i \bigotimes I_{r^i}$ 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.

As shown in [13, 19] the generator matrix of SGSPNs is given by:

$$Q = \bigoplus_{i=0}^{N-1} Q_l^i + \sum_{t \in TS} w(t) \bigotimes_{i=0}^{N-1} Q_t^i - D \tag{1}$$

where D is a diagonal matrix providing row sums of $\bigoplus_{i=0}^{N-1} Q_l^t + \sum_{t \in TS} w(t) \bigotimes Q_t^i$. For $\mathcal{RS}$ exploration the structured representation of Q gives a valid state transition matrix for all states reachable from M_0. For other application areas additional restrictions might apply, e.g. for performance analysis TRG^i need to be strongly connected for ergodicity of the associated CTMC [13]. Based on Q we can reformulate the task of $\mathcal{RS}$ exploration: calculate the minimal, reflexive, and transitive closure of relation $reach(M_x, M_y)$ where $reach(M_0, M_0)$ and

$$reach(M_x, M_y) \iff Q(x, y) \neq 0.$$

Note that due to the elimination of vanishing states during generation of matrices Q^i, $reach(M_x, M_y)$ gives $\mathcal{TRS}$. Obviously $\mathcal{TRS} = \mathcal{RS}$ if $T_i = \emptyset$, hence computation of $\mathcal{RS}$ for untimed Place/Transition nets is simply possible by interpreting such nets as GSPNs where all transitions are timed. In this sense a strictly speaking $\mathcal{TRS}$ exploration algorithm serves also as a $\mathcal{RS}$ algorithm, such that we subsume such algorithms by "$\mathcal{RS}$ exploration algorithms" in the following. Before we describe such an algorithm, we clarify similarities to stochastic automata networks (SANs) [20].

A Side-glance on SANs On the level of component state spaces, a matrix Q^i can be seen as a matrix representation of a finite automaton i with additional edge labels specifying a Markovian timing. The set of component matrices can also be interpreted as a set of automata with synchronous communication. SANs follow this point of view and their structured representation coincides with Eq. 1, such that we can conclude that the RS algorithm given in Sec. 3 also applies to SANs. For more information on SANs the interested reader is referred to [20, 21, 22]. SANs have been analyzed in continuous and discrete time, here we consider continuous time.

Definition 5. A SAN consists of N stochastic automata (SAs) with index set $IS = \{0, \ldots, N-1\}$ such that SAs are numbered consecutively from 0 to $N-1$. Every automaton i is characterized by its finite state space $\mathcal{RS}^i$ containing k^i states and its transition function. States in $\mathcal{RS}^i$ are numbered consecutively form 0 to $k^i - 1$ starting from the initial state. All timing is Markovian. The following types of transitions are possible: local transitions which occur locally in $\mathcal{RS}^i$ without affecting other automata, synchronized transitions which have to occur synchronously in a set of automata, and functional transitions, where the transition rate is a nonnegative, real-valued function of the state of other automata. Local and synchronized transitions can be functional. Transitions which have a fixed rate are denoted as constant. A SAN includes TS different synchronized transitions (events); a subset of at least two automata in IS participates on a synchronization event $t \in TS$.

It is straightforward to show that the complete SAN specifies a CTMC, if the single automata observe Markovian timing. The complete CTMC can be described as a N-dimensional CTMC with state space $\mathcal{RS} \subseteq \mathcal{PS} := \mathcal{RS}^{N-1} \times \ldots \times \mathcal{RS}^1 \times \mathcal{RS}^0$. A single SA of a SAN is also referred to as a component. We assume for functional transitions, that their functions do not interfere with the logical behavior, i.e. the function does not evaluate to zero if the transition can occur. This means that the function determines the delay, but not the fact that the transition can occur. Hence the set of reachable states for a given SAN is independent of the selection of functions, such that we can safely replace any functional transition by a constant transition during state space exploration. Hence for $\mathcal{RS}$ analysis tensor operations of Def. 4 are sufficient and it is not necessary to use generalized tensor operations [22].

For the definition of a structured representation of generator matrix Q for SANs we start with the definition of some matrices considering a single automa-

ton. Let A^i denote that a matrix A belongs to automaton i. Any such A^i is a $k^i \times k^i$ matrix. Let Q_l^i be a matrix containing local transition rates. For every synchronizing event t define Q_t^i as the transition matrix of automaton i containing transition rates for t. Every matrix Q_t^i contains only nonnegative elements. For automata that do not participate in event t we define $Q_t^i = I_{k^i}$. Usually one of the matrices belonging to event t contains the corresponding transition rates, all others have row sums equal to 0 or 1, depending whether event t is possible or not in the corresponding state. We additionally define $k^i \times k^i$ diagonal matrices $D_l^i = diag(Q_l^i e^T)$ and $D_t^i = diag(Q_t^i e^T)$ containing the row sums of the corresponding rows of Q_l^i and Q_t^i in the main diagonal, e is a row vector of appropriate size with all elements equal to 1. With these matrices the generator matrix Q of the SAN is described as follows:

$$Q = \bigoplus_{i=0}^{N-1} Q_l^i + \sum_{t \in TS} \bigotimes_{i=0}^{N-1} Q_t^i - D \tag{2}$$

where $D := \bigoplus_{i=0}^{N-1} D_l^i + \sum_{t \in TS} \bigotimes_{i=0}^{N-1} D_t^i$. Eq. (2) is a slightly less elegant, but equivalent formulation of the structured representation given in [20, 22]. Obviously the structured representations of SGSPNs and SANs formally coincide if $w(t)$ is multiplied into the first term in the tensor product $\bigotimes Q_t^i$ for SGSPNs, such that we can in the following consider the $\mathcal{RS}$ problem on matrix level without distinguishing between SGSPNs and SANs.

3 $\mathcal{RS}$ Exploration Based on Structured Representations

Assume a structured representation is given for a certain model with initial state M_0. In this section we describe how successor states are calculated from the structured representation and formulate a search algorithm to compute $\mathcal{RS}$ (or $\mathcal{TRS}$ in case of SGSPNs with $T_i \neq \emptyset$). Since diagonal values D in Q are irrelevant for $\mathcal{RS}$ exploration we focus on state transition matrix $\bar{Q} := Q + D$ and let $\bar{Q}_l = \bigoplus_{i=0}^{N-1} Q_l^i$.

Successor states can be reached due to local or synchronized transitions. Considering local transitions, Def. 4 ensures that $\bar{Q}_l = \sum_{i=0}^{N-1} I_{l^i} \otimes Q_l^i \otimes I_{r^i}$ where $l^i = \prod_{j=i+1}^{N-1} k^j$ and $r^j = \prod_{j=0}^{i} k^j$. In consequence state transitions from a state M_x to a state M_y due to a local transition $t \in T^i \backslash TS$ in a component i are all specified in a single term $\bar{Q}_l^i := I_{l^i} \otimes Q_l^i \otimes I_{r^i}$ such that $\bar{Q}_l = \sum_{i=0}^{N-1} \bar{Q}_l^i$ and $M_x[t\sigma > M_y$ with a (possibly empty)firing sequence of immediate transitions σ if and only if $\bar{Q}_l(x,y) \neq 0$. Each term $\bar{Q}_l^i$ specifies a set of state transitions and the sum $\bar{Q}_l = \sum \bar{Q}_l^i$ behaves like a logical OR since all entries are nonnegative[1]. According to Def. 4 nonzero entries in $\bar{Q}_l$ are characterized by

$$\bar{Q}_l(x,y) \neq 0 \iff \exists i \in IS: \; Q_l^i(x^i, y^i) \neq 0 \wedge \forall j \in IS, j \neq i : x^j = y^j \tag{3}$$

[1] In fact matrices Q_l^i, Q_t^i can also be mapped to boolean matrices and $+, *$ to $\wedge, \vee$ for $\mathcal{RS}$ analysis.

For a synchronized transition t let $IC(t) := \{i | i \in IS \wedge t \in T^i\}$ denote the set of involved components, then $\bar{Q}_t := \bigotimes_{i=0}^{N-1} Q_t^i$ follows with a similar argumentation as for (3):

$$\bar{Q}_t(x,y) \neq 0 \iff \forall i \in IC(t) : Q_t^i(x^i, y^i) \neq 0 \wedge \forall j \notin IC(t) : x^j = y^j \quad (4)$$

This follows from Def. 4, plus the fact that in general $\prod a_i \neq 0 \iff \forall i : a_i \neq 0$, and $Q_t^j = I \; \forall j \notin IC(t)$. 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 as follows: in case of local transitions the value y for a $\bar{Q}_l(x,y) \neq 0$ is given by y = mix($x^{N-1}, \ldots, x^{i+1}, y^i, x^{i-1}, \ldots, x^0$). 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 Q_l^i, resp. Q_t^i, we can store Q_l^i with $Q_l^i(x^i, (y^i - x^i) \cdot g_i)$ resp. $Q_t^i(x^i, (y^i - x^i) \cdot g_i)$ on the same place instead, such that calculation of a successor state requires $|IC(t)|$ additions and no multiplications.

The basic algorithm follows the standard search algorithm for state space exploration by traversing the reachability graph, e.g. [8], but the calculation of successor markings is adapted to the context of structured representations:

Input: Matrices of a structured representation
Program:
Init: $\mathcal{RS} = \{M_0\}$, 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 $Q_l^i(M_x^i, M_y^i) \neq 0$
        $M_y = M_x + (M_y^i - M_x^i) \cdot g_i$
        if $M_y \notin \mathcal{RS}$
        then insert M_y in $\mathcal{RS}$ and S
    foreach $t \in TS$
      if $\forall i \in IC(t) : \exists Q_t^i(M_x^i, M_y^i) \neq 0$
(*)   then foreach combination of elements $Q_t^i(M_x^i, M_y^i) \neq 0$ over IC(t)
        $M_y = M_x + \sum_{i \in IC} (M_y^i - M_x^i) \cdot g_i$
        if $M_y \notin \mathcal{RS}$
        then insert M_y in $\mathcal{RS}$ and S
end

In line (*) only one combination occurs if each of the corresponding rows $Q_t^i(M_x^i, .)$ of IC(t) contains exactly one nonzero entry. Due to subsequently firing immediate transitions, several nonzero entries per row are possible, such that for the

general case one has to consider all combinations of such nonzero entries to derive all successor states.

Correctness follows from the fact, that the conventional method [8] is employed and only calculation of successor states (according to Eqs (3) and (4))and data structures are especially adapted. Successor states are obtained from the state transition matrix part of Q. Coding and decoding of states into vectors of component states follows mix, resp mix^{-1}, which coincides with the tensor products in the structured representation of Q.

It is well known, that the crux of the conventional method is the choice of appropriate data structures for $\mathcal{RS}$ and S. Conflicting interests are that due to the number of tests, member and insert operations for $\mathcal{RS}$ have to be as fast as possible but on the other hand due to the size of $\mathcal{RS}$ the memory spent per element of $\mathcal{RS}$ must be minimized to keep the method applicable for large state spaces. Additionally the size of $\mathcal{RS}$ is unknown in the general case. In the context of structured representations $\mathcal{RS}$ is a subset of $\mathcal{PS} = \times_{i \in IS} \mathcal{RS}^i$. We suggest to use hashing and a hash table v of boolean entries with $v(x) = true$ if $M_x \in \mathcal{RS}$ and $v(x) = false$ otherwise. Since $mix : \mathcal{PS} \rightarrow \{0, 1, \ldots, k-1\}$ is bijective, it gives a perfect hash function, no collisions can occur by $x = mix(M_x)$. This results in a bit-vector of length $\mathcal{PS}$ with 1 bit per state for v. Member and insert operations for $\mathcal{RS}$ are in O(1), since the costs for evaluating mix are already included in the computation of successor states. Due to function mix the set S can be represented by a stack which contains just integer values. This is memory efficient compared to storing vectors of component states or markings. Push and pop operations on stacks are in O(1). Decoding of a state into component states takes N division and modulo operations. Coding of component states in the calculation of a successor markings due to firing of a transition requires as many additions as components are involved (at most IC(t)) as discussed above.

The length of the bitvector to represent $\mathcal{RS}$ can be extreme if $|\mathcal{PS}| >> |\mathcal{RS}|$. Compared to other representations of $\mathcal{RS}$ note, that typically the memory for 1 pointer or 1 integer is about 32 bits for state representations based on marking vectors. If coding techniques and tree-type data structures according to [8] are used the amount of memory used for $\mathcal{RS}$ depends on the number of places, the maximum number of tokens estimated for each place and the reachability relation itself, such that a comparison is difficult to draw in general. In the context of bitstate hashing, Holzmann [17] assumes 64 bits per state in case of the hashcompact method [26] or allows for about 100 bits per state for an optimal working area of the double bit hashing method; for values less than 100 bits per state double bit hashing gradually looses coverage, i.e. the quality of results is reduced. This is different in our approach: mix gives a perfect hash function, such that the new algorithm is exact. In summary we can conclude that compared to any alternative state representation which uses at least 64 bits per state, the bitvector representation used here requires less(!) memory for $\mathcal{RS}$, if $|\mathcal{RS}|/|\mathcal{PS}| \geq 1/64 \approx 1.5\%$. Surely for a large set of models holds that 1.5% of $\mathcal{PS}$ is reachable. As a rule of thumb, a bitvector representation is applicable if it fits into the available main memory of a given hardware configuration, e.g. on

a 32 MB machine $\mathcal{PS}$ can be up to 256 million states. For larger values of $\mathcal{PS}$ it is model dependent, because for a lot of models *mix* implies a certain locality which goes well with the locality assumption of virtual memory concepts.

4 Improvements for state space exploration

Interleaving of independent, concurrent transitions has been recognized as one source of the state space explosion problem and as a source for computational overhead during state space exploration, e.g. during elimination of vanishing markings the exploitation of extended conflict sets (ECS) can yield significant savings in computation time by avoiding unnecessary interleavings [1, 2]. In this context priorities are assigned to independent sets of transitions in order to reduce the degree of concurrency, i.e. to reduce the number of firing sequences taken into consideration. A more general approach in state space exploration is given by Valmari [23, 24] with the stubborn set method; the point of interest is, whether certain states are reachable or not. From this point of view interleavings caused by independent, concurrent transitions introduce a lot of "redundant" states which can safely be omitted, i.e. $\mathcal{RS}$ is reduced.

In our context the goal is to consider all nodes of RG but not all arcs. A spanning tree on RG would be ideal, however here we establish only a way to omit certain arcs of RG. In SGSPNs and SANs[2] the partition into components implies that transitions which are local and belong to different components are independent of each other. Interleaving of independent, concurrent transitions has the effect that a lot of states are reached over and over again if a straightforward search algorithm is employed. However it is not necessary to consider all permutations of these independent transitions. A selection of just one ordered sequence is sufficient if it is applied on sets of states. The example in Fig. 1 illustrates the idea by a net with components A, B, and C which have only local transitions. Not all arcs of the RG in Fig. 1 b) have to be considered for $\mathcal{RS}$ exploration. One method is to order the local transitions by components and apply them on sets of states. We start from state abc, apply the transition of component A and derive a'bc. As a second step the transition of component B is applied on set { abc, a'bc }, since all states in this set have the same component state for component B. This yields the set { abc, a'bc, ab'c, a'b'c}. Finally on all of these states we apply the local transition of component C with local state c. This procedure considers for the reachability graph of Fig. 1 b) the graph of Fig. 1 c) instead, where the number of arcs is significantly reduced. The method gains in efficiency if not only one local transition per component is fired as in Fig. 1 but sequences of local transitions per single component. This requires an additional termination criterion for such a local search. A natural criterion inherited from the conventional search method is to stop a local search when the global state is already element of $\mathcal{RS}$, e.g. if a'bc is already in $\mathcal{RS}$, no successors { a'b'c, a'b'c', a'bc'} would be considered in this search. This results in a massive pruning of such a search. Validity of this approach is proved later in the paper. Extending the search of local successors to sets of states allows that only some

[2] In fact this is typical for all modular models which communicate via fused transitions.

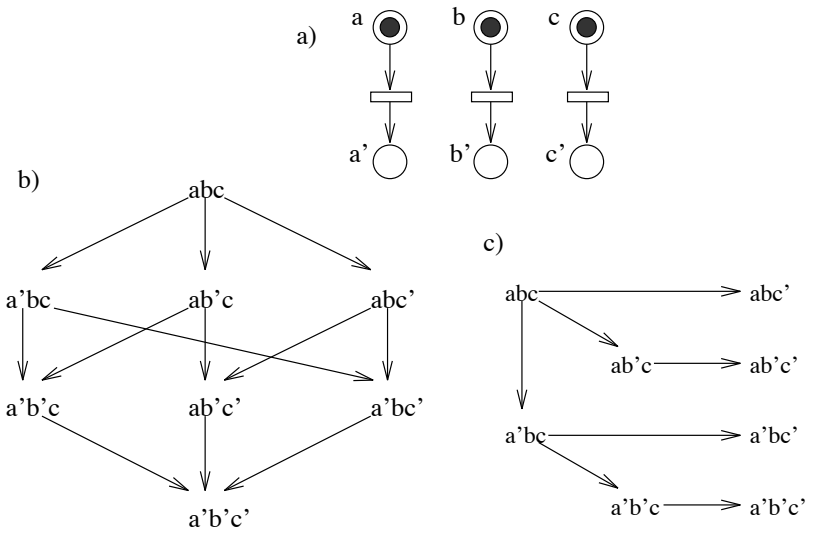

Fig. 1. An example net with 3 components, its reachability graph RG and the considered subgraph of RG

states require a complete decoding with N division and modulo operations, some states require only a partial decoding and some states require only a decoding for one component in the following algorithm. The basic idea is to keep track of the differences, resp. similarities between states and their successor states.

Input: Matrices of a structured representation
Program:
Init: x = 0 ; $\mathcal{RS}_x = \{M_0\}$; $S_x = \{M_0\}$
begin
  while not empty S_x
    take M_x out of S_x (step x)
    start = stop = last = 1
    copy state M_x to array(1)
    decode M_x into $(M_x^{N-1}, \ldots, M_x^0)$
    check TS on M_x like check-TS(0,last) but without extra decoding
    foreach component $i \in IS$ in increasing order
      foreach array element M between start and stop
        start-TS = last + 1
        search(i,M_x^i,M)
        check-TS(i,start-TS)
      stop = last (last element in array)
    $\mathcal{RS}_{x+1} = \mathcal{RS}_x; S_{x+1} = S_x; x = x + 1;$
end

search(i,M_x^i,M)
    foreach M_y^i : $Q_l^i(M_x^i, M_y^i) \neq 0$
        $M' = M + (M_y^i - M_x^i) \cdot g_i$
        if $M' \notin \mathcal{RS}_x$
        then last = last + 1
            array(last) = M'
            $\mathcal{RS}_x = \mathcal{RS}_x \cup \{M'\}$
            search(i,M_y^i,M')

check-TS(h,start-TS)
    pick one array element M between start-TS and last
    decode M into M^j for $j < h$ (exploit $M_x^j = M^j$ for $j > h$)
    foreach array element $\bar{M}$ between start-TS and last
        decode $\bar{M}$ only at position h and update $M^h = \bar{M}^h$
        foreach $t \in TS$
            if $\forall i \in IC(t) : \exists Q_t^i(M^i, M_y^i) \neq 0$
            then foreach combination of nonzero elements $Q_t^i(M^i, M_y^i)$ over IC(t)
                $M' = M + \sum_{i \in IC}(M_y^i - M^i) \cdot g_i$
                if $M' \notin \mathcal{RS}_x$
                then insert M' in $\mathcal{RS}_x$ and S_x

Compared to the algorithm of Sec. 3, local transitions are only considered by a component specific call for function search, which stores new states in a special array, while synchronized transitions are only considered in function check-TS, which stores new states in stack S_x as before. Apart from the sophisticated decoding, function check-TS corresponds to the 2nd foreach-loop in the previous algorithm. A call for function search causes a component specific depth-first-search (DFS) via local transitions whose termination depends on $\mathcal{RS}_x$. Since array entries are caused by local transitions and new states are inserted behind position stop, all entries between positions start and stop have equivalent values M_x^i when component i is considered. A byproduct of this calculation is that a set of successor states is successively stored in the array (between positions check-TS and last), which differs only in component i, hence this structural information can be exploited to speed up decoding of states and checking synchronized transitions. In summary this technique avoids unnecessary interleavings, saves division and modulo operations for decoding, and simplifies enabling tests for synchronized transitions; surely these effects occur only if the model under consideration allows for a sufficient amount of independent, local transition firings.

Termination follows from the observation that any state is inserted at most once into the array or S_x. To ensure that the algorithm actually considers all reachable states, we have to show that any successor of a state in RS_x is either already element of RS_x or it still has a predecessor on stack S_x:

Lemma 6. *After every step x holds:* $\forall M \in \mathcal{RS}_x \backslash S_x$: *if* $\exists t \in T : M[t > M' \wedge M' \notin \mathcal{RS}_x$ *then* $\exists M'' \in S_x, \sigma \in T^* : M''[\sigma > M'$.

Proof. Proof by induction, initially $\mathcal{RS}_0 = S_0 = \{M_0\}$ so lemma is trivially fulfilled by $\mathcal{RS}_x \backslash S_x = \emptyset$.
Induction step: indirect proof by contradiction, assume lemma does hold for step x but does not hold in step x+1. In this case for any counterexample $M \in \mathcal{RS}_{x+1} \backslash S_{x+1}$ holds $M_x[\sigma_x > M$ so we choose a suitable M with shortest σ_x.

case $M_x = M$ Since all transitions enabled by M_x are considered by the algorithm $\nexists t \in T : M[t > M'$ such that $M' \notin \mathcal{RS}_{x+1}$.

case $M \in \mathcal{RS}_{x+1} \backslash (S_{x+1} \cup \{M_x\})$ i.e. M was reached in step x+1 for the first time and inserted into $\mathcal{RS}_{x+1}$ but not in S_{x+1}. Consider the kind of transition t_M by which M was reached: if $t_M \in TS$ then according to the algorithm any marking reached for the first time is inserted into $\mathcal{RS}_{x+1}$ and S_{x+1} which contradicts the above assumption, hence $t_M \notin TS$. So $t_M \in T^i \backslash TS$ is local to a certain component $i \in IS$. Since M is reached for the first time searching from M_x, it is reached by the search algorithm following a sequence $\sigma_M \in (T \backslash TS)^* : M_x[\sigma_M > M$ of local transitions in the order of components, so $\sigma_M = \sigma_0 \cdot \sigma_1 \ldots \sigma_{N-1}$ with $\sigma_i \in (T^i \backslash TS)^*$ forall $i \in IS$.
Consider now the critical transition t for $M[t > M'$: $t \in TS$ is not possible, because M was reached for the first time, so according to the algorithm all synchronized transitions are considered for M. So $t \in T^j \backslash TS$ must be local to some component $j \in I$.
If $i < j$ then $t \in T^j \backslash TS$ is considered within a DFS-search for M, since M is stored in the array as any new element reached by local transitions and a search is called as for all elements between start and stop. In consequence M' would be element of $\mathcal{RS}_{x+1}$.
If $i = j$ then M is reached within a DFS-search of the same component i=j. According to the algorithm every $t \in T^i \backslash TS = T^j \backslash TS$ is considered such that M' would be reached and inserted into $\mathcal{RS}_{x+1}$.
If $i > j$ then $M_x[\sigma_M t > M'$ and the firing sequence does not follow the order of I due to transition t. Due to the independence of the firing of local transitions which belong to different components, firing sequence $\sigma_M t$ has permutation $\sigma_{perm} = \sigma_1 \cdots \sigma_j t \sigma_{j+1} \ldots \sigma_N$ which can be fired and yields M' as well. According to our assumption $M_x \in \mathcal{RS}_{x+1}$ and $M' \notin \mathcal{RS}_{x+1}$, hence this firing sequence has a first break where $M_1[t_1 > M_2$ and $M_1 \in \mathcal{RS}_{x+1}$ and $M_2 \notin \mathcal{RS}_{x+1}$. Let σ_1 denote the firing sequence reaching M_1 being a prefix of σ_{perm}. Since $t_1 \in T \backslash TS$ must be a local transition and M_2 is not reached by a DFS-search, DFS must terminate at M_1 because $M_1 \in \mathcal{RS}_{x+1}$ holds already. There are 2 possibilities:
Firstly $M_1 \in \mathcal{RS}_x$ was reached in one of the previous steps. Then due to $M_2 \notin \mathcal{RS}_x \subseteq \mathcal{RS}_{x+1}$ either $M_1 \in S_x$ or $\exists M'' \in S_x, \sigma_2 \in T^* : M''[\sigma_2 > M_2$.
If $M_1 \in S_x$, then due to $M_2 \notin \mathcal{RS}_{x+1}$ must be $M_x \neq M_1$. Hence $M_1 \in S_{x+1}$, which in turn implies M_1 is a suitable element in $S_{x+1}, \exists \sigma_1 \in T^* : M_1[\sigma_1 > M'$.
Otherwise if $\exists M'' \in S_x, \sigma_2 \in T^* : M''[\sigma_2 > M_2$ and if $M'' = M_x$ is the only suitable element of S_x from which M_2 can be reached then M_1 is already a suitable element for a counterexample like M_x but with a shorter sequence than σ contradicting our assumption. Hence $\exists M'' \neq M_x \in S_x, \sigma_2 \in T^* : M''[\sigma_2 > M_2$.
Since M_x is the only element removed from S_x in step x+1, M'' is element of

S_{x+1} as well. Finally the first break cannot be at M, since this is ensured already above by $M_x \neq M$.
The second possibility is that $M_1 \in \mathcal{RS}_{x+1} \backslash \mathcal{RS}_x$, i.e. M_1 was reached before but within step x+1. Then M_1 is itself a starting point for DFS and t_1 follows in the order of components ($\sigma_1 t$ is an ordered sequence), the transition t_1 is checked at $M_1[t>M_2$. □

Correctness of the algorithm simply follows from the lemma above in combination with the termination criterion $S_x = \emptyset$.

Breadth-First-Search (BFS) can be used instead of DFS for the price of an additional decoding at one component. The main advantage from a practical point of view is that the array entries can be reused as a BFS-queue such that no extra memory is required and recursion as in DFS can be avoided.

5 Examples

As examples for SGSPNs we consider the "benchnet model" of Caselli et al. [6, 7] and the flexible manufacturing system given by Ciardo et al. [10].
The Benchnet model This model has a simple net structure shown in Fig. 2, whose obvious symmetries are ignored by the implemented approach. The size of the state space is increased by modifying arc weights, i.e. by increasing values for parameter k. The net is partitioned into 4 components to obtain a SGSPN, namely A,B,C, and D as denoted in Fig. 2. Finiteness for the state space of component A is achieved by exploiting P-invariants [19], i.e. a P-invariant of the SGSPN implies that p_0 contains at most 3 tokens. Caselli et al. state that main memory is the main bottleneck for state space exploration. Since the SGSPN approach does not solve the state space explosion problem, but reliefs its impacts, it is interesting to see how far state space exploration can be pushed on a given hardware configuration. Figure 3 shows the CPU-time and the elapsed time in seconds for the basic and the improved method as a function of $|\mathcal{RS}|$; the CPU-time does not differ significantly from the elapsed time for $|\mathcal{RS}| \leq 43$ million states. These results are obtained on a Sparc 5 with 70 MHz CPU and 32 MB main memory.

Fraction $\mathcal{RS}/\mathcal{PS} = 25\%$ is constant for all values of k in this model; nevertheless it shows clearly that $|\mathcal{PS}| >> |\mathcal{RS}|$. The computation times for the component state spaces are negligible, i.e. they are less than 1 sec for the state spaces considered here. Component state spaces have very moderate cardinalities, i.e. $|\mathcal{RS}^A| = 4$ and $|\mathcal{RS}^B| = |\mathcal{RS}^C| = |\mathcal{RS}^D| = k + 2$. The number of nonzero entries which have to be stored for the structured representation remains less than 1600 for $k \leq 498$. The benchnet model is a kind of best case example for the improved method because large values of k imply a high degree of enabling for local transitions, such that a lot of interleaving occurs, which in turn is successfully treated by the improved method. The results clearly indicate that the improved method is much more efficient than the basic method for this example. Compared to computation times given in [7] note that the conventional approach in a sequential implementation fails for $|\mathcal{RS}| > 10^6$ on a Sparc 2 with 64 MB main memory; computation times for a parallel implementation on a

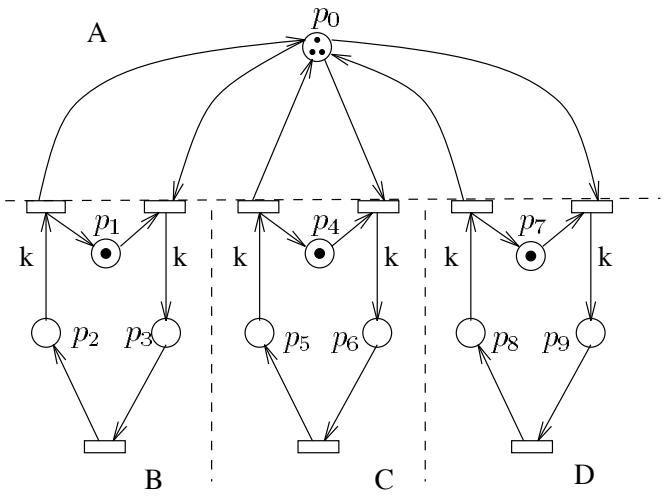

Fig. 2. The benchnet model with a partition into components A,B,C, and D

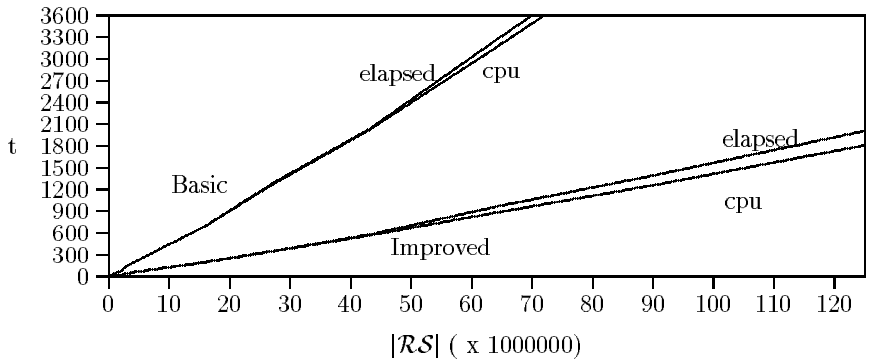

Fig. 3. Computation time t in seconds for benchnet model up to 125 million states

CM-5 are similar to the improved method up to the state space cardinalities considered in [7], i.e. up to 1.7 million states.

Comparisons with results in [6, 7] should be drawn carefully, since the new algorithm relies on the partition into components inherent to SGSPNs and SANs, but the conventional method (sequential or parallel) for GSPNs can be applied on arbitrary GSPNs without such an information. Hence we only conclude that for this example the SGSPN method allows to outperform the conventional sequential approach by far and furthermore to outweigh advantages obtained from parallel computation. Since the new algorithm is amenable to parallelization at

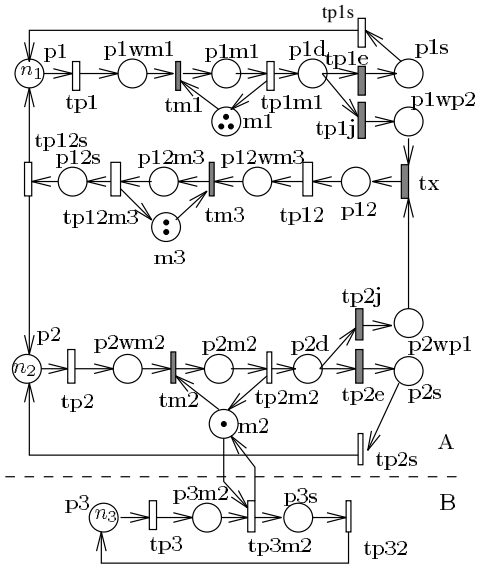

Fig. 4. FMS with 2 components

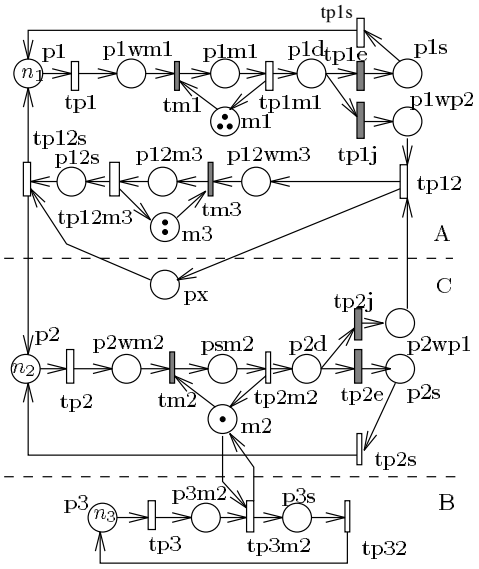

Fig. 5. FMS with 3 components

least as much as the conventional state space exploration, further improvements are foreseen.

A Flexible Manufacturing System Ciardo et al. [10] describe a flexible manufacturing system (FMS) to discuss the benefits of an approximate analysis technique based on decomposition. Figure 4 models this FMS as given in [10], it consists of three machines m_1,m_2, and m_3. Machine m_1 processes parts of type p_1, up to three at a time as $M_0(m_1) = 3$ indicates. Machines $m2$ and $m3$ are modeled similarly. Machine m_2 processes parts of type p_2 or p_3, but parts of type p_2 have a higher priority than p_3. Machine m_3 assembles parts p_1 and p_2 into a new type p_{12}. Finished parts p_1,p_2,p_3, or p_{12} can be shipped and in this case the same number of rough parts enters the system, to maintain a constant inventory. Since we are only interested in $\mathcal{RS}$ exploration here, we omit the additional assumptions considering aspects of time in [10] apart from the fact that $T_i := \{tm1, tm2, tm3, tp1, tp2, tp1j, tp2e\}$ are immediate transitions and all other transitions are timed. In [10] "flushing" arcs connect corresponding transitions tp1s,tp2s,tp3s, and tp12s; they have a marking dependent cardinality equal to the number of tokens in the input place for the transition. This exceeds the GSPN definition underlying the SGSPNs introduced here. For simplicity we assume constant arc weights for transitions tp1s,tp2s,tp3s, and tp12s. The FMS model is exercised for 2 different partitions:

FMS with partition into 2 components A decomposition into 2 components by separating processing of parts of type p_1,p_2 and p_{12} from parts p_3. In this model transition tp3m2 is the only synchronized transition. Table 1 shows the corre-

N	TRS^A	TRS^B	$\mathcal{PS} = \mathcal{TRS}$	$\sum NZ(Q^i)$	Comp-expl. in sec	TRS-expl. in sec
3	652	10	6520	3391	2	2
4	2394	15	35910	14088	5	6
5	7272	21	152712	46719	19	22
6	19206	28	537768	131769	66	77
7	45540	36	1639440	328830	259	296

Table 1. Results of FMS model with partition into 2 components

sponding results for increasing values of $N = n_1 = n_2 = n_3$: the columns $\mathcal{TRS}^A$ and $\mathcal{TRS}^B$ give the sizes of component state spaces. $\mathcal{TRS}^A$ contains processing of parts p_1,p_2, and p_{12}; TRS^B processing of parts p_3. Column four shows the size of $\mathcal{PS}$ which equals $\mathcal{TRS}$ for this partition. Column $\sum NZ(Q^i)$ gives the number of matrix elements which are explicitly stored in the structured representation. Column six gives the elapsed time used for the state space exploration of components; the total elapsed time in seconds for state space exploration using the improved method is given in column seven, note that the values include values of column six.

Obviously the synchronization of components A and B via transition tp3m2 does not restrict reachability, all elements of the cross-product of component state spaces are reachable in this model. As a consequence the TRS-exploration of the SGSPN can safely be omitted for analysis purposes, nevertheless it is an interesting extreme case, in which the hash table is completely filled. The results demonstrate that most of the time in state space exploration is used to explore the state space of component A which uses the conventional approach. Once generation of component state spaces is finished, exploration of the overall TRS is extremely fast, e.g. 296-259=37 seconds for exploration of a TRS with more than 1.6 million states. Trivially, the implementation with hashing into a bit vector using a perfect hash function is insensitive to the filling of the hash table.

Since the dimension of component state spaces are of different orders of magnitude, such that their generation times differ significantly, a further decomposition of component A seems to be worthwhile.

FMS with partition into 3 components Component A of this model can be split into 2 components if transition tx is merged with transition tp12. The new model is partitioned into three components A,B,C as shown in Fig. 5, where transitions tp12s, tp12 and tp3m2 are synchronized transitions. Finiteness of TRS^C is ensured by taking place capacities/limits into consideration which follow from P-invariants of the complete model [19]. Nevertheless introducing an additional place px derived from a P-invariant is helpful to avoid useless states in $\mathcal{TRS}^C$. Informally px aggregates the processing of p12. Result values are given in Table 2 and Fig. 6. Note that for this partition $|\mathcal{PS}| >> |\mathcal{TRS}|$ and the fraction $\mathcal{TRS}/\mathcal{PS}$ is rapidly decreasing for increasing values of N.

The results demonstrate that the time for state space exploration is signif-

N	TRS^A	TRS^B	TRS^C	$\mathcal{PS}$ (x 1000)	$\mathcal{TRS}$ (x 1000)	$\mathcal{TRS}/\mathcal{PS}$	$\sum NZ(Q^s)$
3	56	10	35	19	6	0.3327	289
4	126	15	70	132	35	0.2714	697
5	252	21	126	666	152	0.2290	1458
6	462	28	210	2716	537	0.1980	2758
7	792	36	330	9408	1639	0.1742	4838
8	1287	45	495	28667	4459	0.1556	8001
9	2002	55	715	78728	11058	0.1405	12619
10	3003	66	1001	198396	25397	0.1280	19140
11	4368	78	1365	465060	54682	0.1176	28095

Table 2. Results of FMS model with partition into 3 components

icantly reduced if component state spaces are chosen to be small, e.g. for N=7 the elapsed time for the generation of component state spaces is 1 second instead of 259 for the model with 2 components and the complete $\mathcal{TRS}$ exploration requires 68 (52) seconds for the basic (improved) algorithm instead of 296 before. On the other hand it is quite clear that components should not be decomposed arbitrarily fine since their "local" behavior is one source of an efficient TRS-exploration, cf. Sec. 4 and $|PS| >> |TRS|$ is the price paid for superposition. For N=11 the bitvector requires about 58 MB and exceeds the available 32 MB by far, such that a significant amount of paging activities increase the elapsed time, i.e. it takes 9109 seconds for the improved method to explore 54 million states while the CPU-time is only 2442 seconds. This effect is well known from the conventional approach and is experienced there for much smaller state spaces.

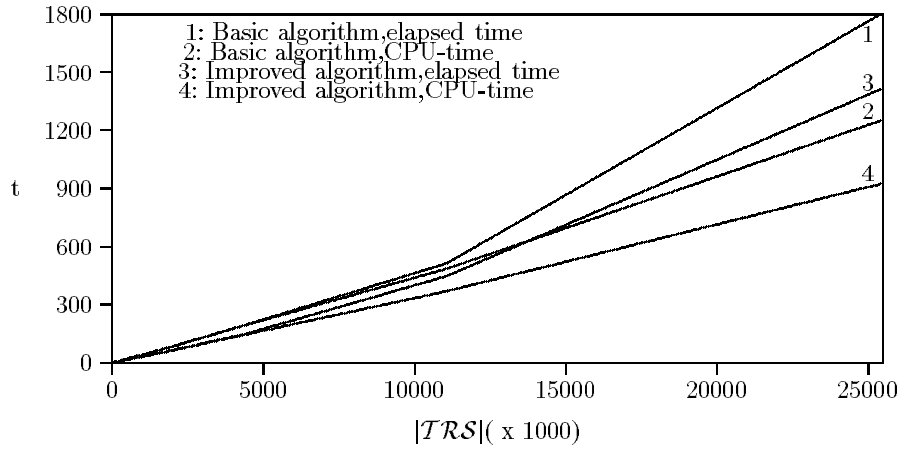

Fig. 6. Computation time t in seconds for FMS model up to 25 million states

6 Conclusions

In this paper a state space exploration algorithm is described, which computes the set of tangible reachable states ($\mathcal{TRS}$) for superposed generalized stochastic Petri nets (SGSPNs) and stochastic automata networks (SANs). Computation of the reachability set ($\mathcal{RS}$) for untimed Place/Transition nets is included as a special case. The algorithm exploits that SANs and SGSPNs by definition provide a decomposition of a model into components and profits from the structured representation known for the generator matrix Q of the associated CTMC. Q is trivially suitable as a state transition matrix for reachability analysis. Additionally the algorithm is based on hashing with a bitvector for the cross-product of component state spaces $\mathcal{PS}$ and a perfect(!) hash function derived from the structured representation. An improvement of this basic algorithm which reduces the impact of interleaving is also presented. The applicability of the new method is demonstrated by two examples for SGSPNs taken from literature [7, 10]. For the given decomposition the new approach outperforms the conventional algorithm in a sequential implementation by far in terms of the size of state spaces and computation times. Both examples show that state space cardinalities of several million states can be handled on a workstation with 32MB main memory.

Although originally designed to improve iterative numerical analysis of SANs and SGSPNs, efficiency of the algorithm motivates its application for any purely functional analysis which is based on state space exploration, e.g. to decide reachability, liveness etc. Since the algorithm does not(!) rely on stochastic timing its application to colored Petri nets with transition fusion [9] is straightforward. In [4, 5] structured representations similar to SANs are given for Markovian process algebras (MPAs), a modeling formalism where a set of processes interact via synchronized transitions as well. The approach given here can be adapted to MPAs as well.

So far the method requires that a decomposition of a model into components with finite component state spaces is given as a prerequisite. Ongoing work is dedicated to clarify the role of P-invariants for the derivation of a suitable decomposition. Another promising perspective is parallelization, since the generation of component state spaces is trivially possible in parallel and the computation of successor states based on component matrices shows at least as much potential for parallelization as the parallel computation of transition firings in GSPNs.

References

1. 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. Work. Petri Nets and Performance Models*. IEEE Computer Society, 1987.
2. A. Blakemore. The cost of eliminating vanishing markings from generalized stochastic Petri nets. In *3rd Int. Work. Petri Nets and Performance Models*. IEEE Computer Society, 1989.
3. P. Buchholz. Numerical solution methods based on structured descriptions of Markovian models. In *5th Int. Conf. Modeling Techniques and Tools*, 1991.

4. P. Buchholz. Markovian process algebra: Composition and equivalence. In *2nd Work. Process Algebras and Performance Modelling*, 1994.
5. P. Buchholz. On a Markovian process algebra. Technical Report 500, Universität Dortmund, 1994.
6. S. Caselli, G. Conte, F. Bonardi, and M. Fontanesi. Experiences on SIMD massively parallel GSPN analysis. In *Computer Performance Eval.*, Springer, 1994.
7. S. Caselli, G. Conte, and P. Marenzoni. Parallel state space exploration for GSPN models. In *16th int. Conf. Application and Theory of Petri Nets*, Springer, 1995.
8. G. Chiola. Compiling techniques for the analysis of stochastic Petri nets. In *4th Int. Conf. on Modeling Techniques and Tools*, Mallorca, 1989.
9. S. Christensen and L. Petrucci. Modular state space analysis of coloured Petri nets. In *16th int. Conf. Application and Theory of Petri Nets*, Springer, 1995.
10. G. Ciardo and K.S. Trivedi. A decomposition approach for stochastic Petri net models. In *4th Int. Work. Petri Nets and Performance Models*. IEEE Computer Society, 1991.
11. M. Davio. Kronecker products and shuffle algebra. *IEEE Transactions on Computers*, C-30(2):116–125, February 1981.
12. S. Donatelli. Superposed stochastic automata: a class of stochastic Petri nets with parallel solution and distributed state space. *Performance Eval.*, 18:21–26, 1993.
13. S. Donatelli. Superposed generalized stochastic Petri nets: definition and efficient solution. In *15th int. Conf. Application and Theory of Petri nets*, Springer, 1994.
14. P. Godefroid. Using partial orders to improve automatic verification methods. In *2nd int. Work. Computer Aided Verification*, LNCS 531. Springer, 1990.
15. P. Godefroid and D. Pirottin. Refining dependencies improves partial-order verification methods. In *5th int. Conf. Computer Aided Verification*, LNCS 697. Springer, 1993.
16. G.J. Holzmann. On limits and possibilities of automated protocol analysis. In *7th int. Work. Protocol Specification, Testing, and Verification*. North-Holland, 1987.
17. G.J. Holzmann. An analysis of bitstate hashing. In *15th int. Symp Protocol Specification, Testing and Verification*, IFIP. Chapman & Hall, 1995.
18. P. Kemper. Closing the gap between classical and tensor based iteration techniques (extended abstract). In *Computations with Markov Chains*. Kluwer, 1995.
19. P. Kemper. Numerical analysis of superposed GSPNs. In *6th Int. Work. Petri Nets and Performance Models*. IEEE Computer Society Press, 1995.
20. B. Plateau and K. Atif. Stochastic automata network for modelling parallel systems. *IEEE Trans. on Software Engineering*, 17(10):1093–1108, 1991.
21. B. Plateau and J.M. Fourneau. A methodology for solving Markov models of parallel systems. *Journal of Parallel and Distributed Computing*, 12, 1991.
22. W.J. Stewart. *Introduction to the numerical solution of Markov chains*. Princeton University Press, 1994.
23. A. Valmari. Error detection by reduced reachability graph generation. In *9th Europ. Work. Application and Theory of Petri Nets*, Venice, Italy, 1988.
24. A. Valmari. Stubborn sets for reduced state space generation. In *Advances in Petri Nets*, LNCS 483. Springer, 1990.
25. P. Wolper and P. Godefroid. Partial order methods for temporal verification. In *4th int. Conf. Concurrency Theory*, LNCS 715. Springer, 1993.
26. P. Wolper and D. Leroy. Reliable hashing without collision detection. In *5th int. Conf. Computer Aided Verification*, Elounda, Greece, 1993.