Advances in compositional approaches based on Kronecker algebra: Application to the study of manufacturing systems

Gianfranco Ciardo
Department of Computer Science College of William and Mary
Williamsburg, VA 23187-8795, USA

Abstract

Recent developments in the use of Kronecker algebra for the solution of continuous-time Markov chains (CTMCs), in particular models based on stochastic Petri nets, have increased the size of the systems that can be analyzed using exact numerical methods. We report on the more recent results, and employ them to study a kanban system.

1 Introduction

Queueing networks, stochastic automata networks, and generalized stochastic Petri nets (GSPNs) are ideally suited to model systems with complex but structured state spaces. The only limitations are the type of timing that can be exactly represented (exponential or phase-type distributions, in the continuous case) and the size of the underlying state space $\mathcal{S}$ (this practical limitation reduces our ability of using phase-type distributions). The size of the transition rate matrix $\mathbf{R}$ for the CTMC is then the main obstacle. On a single workstation, CTMCs with more than 10^6 states and 10^7 nonzero entries in $\mathbf{R}$ are likely to require excessive memory and execution times.

Plateau [6] proposed to describe the matrix $\mathbf{R}$ of a large model by combining a set of much smaller matrices through Kronecker operators. Thus, $\mathbf{R}$ is only implicitly stored and the next larger data structure, usually the steady-state probability vector, becomes the memory bottleneck (we discuss only the steady state solution of ergodic models, but analogous comments apply to the study of sojourn times in transient states up to absorption for non-ergodic chains or to the transient analysis of general CTMCs).

2 Background

We briefly recall the definition of the Kronecker product $\mathbf{A} = \bigotimes_{k=0}^{K-1} \mathbf{A}^k \in \mathbb{R}^{\hat{n} \times \hat{n}}$ of K square matrices $\mathbf{A}^k \in \mathbb{R}^{n_k \times n_k}$, using the convention that the rows and columns of both $\mathbf{A}$ and the matrices $\mathbf{A}^k$ are indexed starting at 0. The generic element $\mathbf{A}_{\mathbf{i},\mathbf{j}}$ is

$$\mathbf{A}_{\mathbf{i},\mathbf{j}} = \mathbf{A}^0_{\mathbf{i}_0,\mathbf{j}_0} \cdots \mathbf{A}^{K-1}_{\mathbf{i}_{K-1},\mathbf{j}_{K-1}}$$

with $0 \le \mathbf{i}_k, \mathbf{j}_k < n_k$, where we use a mixed-base numbering scheme, so that the tuple $\mathbf{i} = (\mathbf{i}_0, \ldots, \mathbf{i}_{K-1})$ corresponds to the number $(\ldots((\mathbf{i}_0)n_1 + \cdots)n_{K-1} + \mathbf{i}_{K-1}$.

An important property of the Kronecker product is that the number of nonzero entries $\mathbf{A}$ is $\eta(\mathbf{A}) = \eta(\mathbf{A}^0) \cdots \eta(\mathbf{A}^{K-1})$, where "$\eta$" indicates the number of nonzero entries in its argument.

The Kronecker sum $\bigoplus_{k=0}^{K-1} \mathbf{A}^k$ is defined as

$$\bigoplus_{k=0}^{K-1} \mathbf{A}^k = \sum_{k=0}^{K-1} \mathbf{I}_{n_0^{k-1}} \otimes \mathbf{A}^k \otimes \mathbf{I}_{n_{k+1}^{K-1}},$$

where $n_i^j = \prod_{k=i}^{j} n_k$.

Most work on compositional approaches focus on GSPNs, but any other high-level formalism can be used, as long as it defines:

- A set of "potential" states: $\hat{\mathcal{S}}$.
- A set of possible events: $\mathcal{E}$.
- An initial state: $\mathbf{i}^0 \in \hat{\mathcal{S}}$.
- Which events are active in a state:
  $\text{active} : \mathcal{E} \times \hat{\mathcal{S}} \to \{True, False\}$.
- The rate of occurrence of an event in a state:
  $\text{rate} : \mathcal{E} \times \hat{\mathcal{S}} \to \mathbb{R}^+$.
- The state reached when an active event occurs:
  $\text{new} : \mathcal{E} \times \hat{\mathcal{S}} \to \hat{\mathcal{S}}$.

These entities are defined compositionally by specifying K submodels with "local" state spaces $\mathcal{S}^k$, so that the potential state space is $\hat{\mathcal{S}} = \mathcal{S}^0 \times \cdots \times \mathcal{S}^{K-1}$. A state $\mathbf{i} \in \hat{\mathcal{S}}$ is then a vector of local states, $\mathbf{i} = (\mathbf{i}_0, \ldots, \mathbf{i}_{K-1})$. The actual state space $\mathcal{S} \subseteq \hat{\mathcal{S}}$ is instead the smallest subset of $\hat{\mathcal{S}}$ containing $\mathbf{i}^0$ and satisfying:

$$\mathbf{i} \in \mathcal{S} \wedge \text{active}(e_l, \mathbf{i}) \wedge \text{new}(e_l, \mathbf{i}) = \mathbf{j} \quad \Rightarrow \quad \mathbf{j} \in \mathcal{S}.$$

The potential state space might be (much) larger than the actual state space $\mathcal{S}$:

$$\hat{n} = |\hat{\mathcal{S}}| = \prod_{k=0}^{K-1} |\mathcal{S}^k| = \prod_{k=0}^{K-1} n_k \geq |\mathcal{S}| = n.$$

An event e_l is local to submodel k iff the value of active, rate, and new for it depends only on component k of the state. Otherwise, it is "synchronizing".

Given a structured approach, the transition rate matrix $\mathbf{R}$ of the CTMC underlying the model is

$$\hat{\mathbf{R}}_{\mathcal{S},\mathcal{S}} = \underbrace{\left(\sum_{l=1}^{L} \lambda_l \bigotimes_{k=1}^{K} \mathbf{W}^{k,l} \right)_{\mathcal{S},\mathcal{S}}}_{\text{synchronizing events}} + \underbrace{\left(\bigoplus_{k=1}^{K} \mathbf{R}^k \right)_{\mathcal{S},\mathcal{S}}}_{\text{local events}} \quad (1)$$

(see the Appendix for a definition of the Kronecker product and sum operators), where:

- $\hat{\mathbf{R}}$ is a $\hat{n} \times \hat{n}$ matrix, which coincides with the actual transition rate matrix $\mathbf{R}$ in the entries corresponding to reachable rows and columns.
- $\mathbf{W}^{k,l}$ is a $n_k \times n_k$ matrix defined as

$$\mathbf{W}^{k,l}_{i,j} = \begin{cases} \text{rate}^k(e_l, i) & \text{if active}^k(e_l, i) = True \\ & \text{and } j = \text{new}^k(e_l, i) \\ 0 & \text{otherwise} \end{cases}$$

(Note how rate, active, and new are specified at the submodel level).

- $\mathbf{R}^k$ are the local transition rate matrices describing the effect of local events:

$$\mathbf{R}^k_{i,j} = \sum_{e_l \in \mathcal{E}^k_L} \lambda_l \cdot \mathbf{W}^{k,l}_{i,j}$$

2.1 Potential vs. actual state space

One problem with the compositional approach is that of unreachable states. This is due to synchronizations and other logical constraints among subsystems, which make some combinations of local states impossible.

A similar problem arises for the transition rate matrices. The matrix $\hat{\mathbf{R}} \in \mathbb{R}^{\hat{n} \times \hat{n}}$ is obtained through Kronecker operators on matrices of size $n_0, \ldots, n_{K-1}$. Only its submatrix relative to actual states, $\hat{\mathbf{R}}_{\mathcal{S},\mathcal{S}}$ is of interest, as it satisfies $\boldsymbol{\pi} \cdot \mathbf{Q} = \mathbf{0}$, where $\boldsymbol{\pi} \in \mathbb{R}^n$ and $\mathbf{Q} = \hat{\mathbf{R}}_{\mathcal{S},\mathcal{S}} - \text{rs}(\hat{\mathbf{R}}_{\mathcal{S},\mathcal{S}})$ are the steady state probability vector and the infinitesimal generator of the underlying CTMC, respectively ($\text{rs}(\mathbf{A})$ is a diagonal "row sum" matrix, $\text{rs}(\mathbf{A})_{i,i} = \sum_j \mathbf{A}_{i,j}$).

2.2 Using the potential state space

The discrepancy between $\hat{\mathcal{S}}$ and $\mathcal{S}$ has been attacked in two ways. Initial proposals have simply used a probability vector of size $\hat{n}$, $\hat{\boldsymbol{\pi}} \in \mathbb{R}^{\hat{n}}$ [1, 4, 6]. The main advantage of this approach is that we can define a simple bijection that reflects lexicographic order, $\hat{\Psi} : \hat{\mathcal{S}} \to \{0, \ldots, |\hat{\mathcal{S}}| - 1\}$: $\hat{\Psi}(\mathbf{i}) = \sum_{k=0}^{K-1} \mathbf{i}_k n_{k+1}^{K-1}$.

By initializing $\hat{\boldsymbol{\pi}}$ so that only reachable states have nonzero initial probability (e.g., by setting $\hat{\boldsymbol{\pi}}_{\mathbf{i}^0} = 1$, and the other entries to zero), and using an iterative method such as Power or Jacobi on the entire matrix $\hat{\mathbf{R}}$, we are guaranteed that, upon convergence, only reachable states have a nonzero probability. Indeed, a state will have probability zero iff it is not reachable, provided the chain is ergodic. Thus, given a structured model, we do not even need to explore the state space $\mathcal{S}$ beforehand. We generate the K local state spaces, compute the steady state vector $\hat{\boldsymbol{\pi}}$, and simply set $\mathcal{S} = \{\mathbf{i} : \hat{\boldsymbol{\pi}}_{\mathbf{i}} > 0\}$. The iteration used is

$$\hat{\boldsymbol{\pi}}^{new} \leftarrow \hat{\boldsymbol{\pi}}^{old} \cdot \hat{\mathbf{R}} \cdot \text{rs}(\hat{\mathbf{R}})^{-1}. \quad (2)$$

In practice, we must test for zero values in $\hat{\boldsymbol{\pi}}^{old}$, both to avoid unnecessary operations and to guard against zero entries in $\text{rs}(\hat{\mathbf{R}})$, which might arise only in correspondence to unreachable states $\mathbf{i} \in \hat{\mathcal{S}} \setminus \mathcal{S}$, if the underlying CTMC is ergodic.

This approach, while simple to implement, can be very expensive in terms of both memory and execution time, since it uses data structures of size $\hat{n}$, which can be arbitrarily larger than n.

2.3 Using the actual state space

Kemper [5] was the first one to address the problem of unreachable states, by showing how to use a probability vector of the size of the actual state space, $\boldsymbol{\pi} \in \mathbb{R}^n$. This is achieved at the expense of a $O(\log n)$ multiplicative execution overhead, incurred because the function $\Psi : \mathcal{S} \to \{0, \ldots, |\mathcal{S}| - 1\}$, is still a bijection, but, given $\mathbf{j} \in \mathcal{S}$, the value $\Psi(\mathbf{j})$ cannot be inferred from $\mathbf{j}$ alone, the actual state space has "holes". Every time entry $\mathbf{R}_{\mathbf{i},\mathbf{j}} > 0$ is encountered, for $\mathbf{i} \in \mathcal{S}$, the index of the entry corresponding to state $\mathbf{j}$ in $\boldsymbol{\pi}$ must be found, using a binary search through the state space $\mathcal{S}$.

Kemper's approach is then preferable from an execution standpoint when the overhead due to testing for zero in the vector $\hat{\boldsymbol{\pi}}$ is greater than that of the logarithmic search (the approaches require the same number of iterations in the numerical method). From a memory standpoint, instead, $\boldsymbol{\pi}$ will never require more memory than $\hat{\boldsymbol{\pi}}$, but using the actual state space forces us to store $\mathcal{S}$ in lexicographic order. While we do not discuss particular techniques, we note that an upper bound on

the memory required for this is $O\left(n \cdot \sum_{k=0}^{K-1} \lceil \log n_k \rceil\right)$ bits.

3 Recent developments

As memory is the scarcest resource, we prefer Kemper's approach based on the actual state space: when n is close or equal to $\hat{n}$, somewhat more memory is required but, in many cases the savings can be enormous. However, [5] still has several limitations that hamper its generality and applicability in practice.

First, while we have described the Kronecker approach for an "abstract CTMC-based formalism", GSPNs have been used in many cases. Strictly speaking, GSPNs define a semi-Markov process with two types of sojourn times for the states: zero (vanishing states, with at least one "immediate" transition enabled) and exponentially distributed (tangible states, only "timed" transitions can be enabled). A CTMC is obtained only after "eliminating" the vanishing states. A modeling limitation common to previous work is that the synchronizing events (transitions) must be timed, and their rate must be constant [4, 5].

We have recently been able to lift the above requirements [2], showing that a model with immediate synchronizations can still be analyzed for steady-state through an embedding that "preserves" (some of) the vanishing states. In [2], we were also able to define a marking-dependent behavior which still allows to apply our solution methods. Timed transitions can have rates of the form $\text{rate}(e_l, \mathbf{i}) = \lambda_l \cdot \prod_{k=0}^{K-1} f_k(\mathbf{i}_k)$, where $\lambda_l \in \mathbb{R}^+$ is a positive constant and $f_k : \mathcal{S}^k \to \mathbb{R}^0$ is a nonnegative function of the local state of model k. An analogous dependency is allowed for the weights (unnormalized probabilities) of immediate transitions.

A second, practical, obstacle to the use of compositional approaches is their large execution time requirements. First of all, only the Power or Jacobi methods have been used. This is because $\hat{\mathbf{R}}$ has a very important property: given $\mathbf{i} \in \mathcal{S}$, $\hat{\mathbf{R}}_{\mathbf{i},\mathbf{j}} > 0$ implies that $\mathbf{j}$ is reachable as well, while, given $\mathbf{i} \in \mathcal{S}$, $\hat{\mathbf{R}}_{\mathbf{j},\mathbf{i}}$ can be positive even when $\mathbf{j}$ is not a reachable state. Hence, accessing $\hat{\mathbf{R}}$ by rows, for the rows corresponding to the reachable states, ensures that only relevant entries are processed, while this is not true when accessing $\hat{\mathbf{R}}$ by columns, as needed by the more efficient Gauss-Seidel method. In other words, the iteration in Eq. (2) using $\mathbf{R}$ as described by Eq. (1) can be performed using a call of the form $ByRows(\boldsymbol{\pi}^{old}, \mathbf{W}^{0,l}, \ldots, \mathbf{W}^{K-1,l}, \boldsymbol{\pi}^{new})$, for each synchronizing event and one of the form $ByRows(\boldsymbol{\pi}^{old}, \mathbf{I}_{n_0}, \ldots, \mathbf{R}^k, \ldots, \mathbf{I}_{n_{K-1}}, \boldsymbol{\pi}^{new})$ for each submodel. In either case, statement 9 of procedure *ByRows* is never executed for states $\mathbf{j} \notin \mathcal{S}$ (Fig. 1).

```
ByRows(in: x, A^0, ..., A^{K-1}; inout: y)
 1. for each i ∈ S
 2.     I ← Ψ(i);
 3.     for each j_0 s.t. A^0_{i_0,j_0} > 0
 4.         a_0 ← A^0_{i_0,j_0};
 5.         for each j_1 s.t. A^1_{i_1,j_1} > 0
 6.             a_1 ← a_0 · A^1_{i_1,j_1};
    ...
 7.             for each j_{K-1} s.t. A^{K-1}_{i_{K-1},j_{K-1}} > 0
 8.                 a_{K-1} ← a_{K-2} · A^{K-1}_{i_{K-1},j_{K-1}};
 9.                 J ← Ψ(j);
10.                 y_J ← y_J + x_I · a_{K-1};
```

Figure 1: Computing $\mathbf{y} \leftarrow \mathbf{y} + \mathbf{x} \cdot \mathbf{A}_{\mathcal{S},\mathcal{S}}$ by rows.

Then, as already mentioned, probability vectors of size n are used at the expense of a multiplicative $\log n$ overhead [5], while traditional methods using sparse storage for $\mathbf{R}$ do not have additional overhead.

We have been able to improve on these problems affecting the execution time in two ways. First, we have shown how the use of a Gauss-Seidel-style iteration is feasible, with little additional overhead per iteration. The main difference between the Gauss-Seidel and Jacobi iterations is that, in the former, the variables of the iteration vector need to be computed in a strict sequential order to ensure that we use the "newest information" when computing a given entry of $\boldsymbol{\pi}^{new}$,

$$\boldsymbol{\pi}_j^{new} \leftarrow \left(\sum_{0 \le i < j} \boldsymbol{\pi}_i^{new} \mathbf{R}_{i,j} + \sum_{j < i < n} \boldsymbol{\pi}_i^{old} \mathbf{R}_{i,j} \right) \cdot \text{rs}(\mathbf{R})_{j,j}.$$

Procedure *ByRows* used to implement a Jacobi iteration, instead, updates all entries of $\boldsymbol{\pi}^{new}$ incrementally. Implementing Gauss-Seidel requires to access the nonzero entries of $\mathbf{R}$ by columns. However, as already mentioned, given a state $\mathbf{i} \in \mathcal{S}$, the nonzero entries in row $\hat{\mathbf{R}}_{\mathbf{i},\hat{\mathcal{S}}}$ are exactly those in row $\mathbf{R}_{\Psi(\mathbf{i}),\mathcal{S}}$, while column $\hat{\mathbf{R}}_{\hat{\mathcal{S}},\mathbf{i}}$ contains the nonzero entries in column $\mathbf{R}_{\mathcal{S},\Psi(\mathbf{i})}$ plus possibly others nonzero entries, but only in correspondence to unreachable states. Procedure *ByColumns* in Fig. 2 computes the entries of $\mathbf{y}$ in sequential order, and the only additional complication with respect to the analogous computation by rows is the need to test whether the state $(\mathbf{i}_0, \ldots, \mathbf{i}_{K-1})$ just built is reachable or not. This is accomplished by statement 10. Procedure *ByRows* has a complexity $O(\eta((\bigotimes_{k=0}^{K-1} \mathbf{A}^k)_{\mathcal{S},\mathcal{S}}) \cdot \log n)$, while the possibility of performing unnecessary iterations on unreachable states increases the complexity of *ByColumns* to

ByColumns(in: $\mathbf{x}, \mathbf{A}^0, \ldots, \mathbf{A}^{K-1}$; inout: $\mathbf{y}$)

```
1.  for each j ∈ S
2.      J ← Ψ(j);
3.      for each i_0 s.t. A^0_{i_0,j_0} > 0
4.          a_0 ← A^0_{i_0 j_0};
5.          for each i_1 s.t. A^1_{i_1,j_1} > 0
6.              a_1 ← a_0 · A^1_{i_1,j_1};
    ...
7.              for each i_{K-1} s.t. A^{K-1}_{i_{K-1},j_{K-1}} > 0
8.                  a_{K-1} ← a_{K-2} · A^{K-1}_{i_{K-1} j_{K-1}};
9.                  I ← Ψ(i_0, ..., i_{K-1});
10.                 if I ≠ null
11.                     y_J ← y_J + x_I · a_{K-1};
```

Figure 2: Computing $\mathbf{y} \leftarrow \mathbf{y} + \mathbf{x} \cdot \mathbf{A}_{\mathcal{S},\mathcal{S}}$ by columns.

$O(\eta((\bigotimes_{k=0}^{K-1} \mathbf{A}^k)_{\hat{\mathcal{S}},\mathcal{S}}) \cdot \log n)$. This difference will be minor in most models, since a "typical state" must have as many nonzero entries in its row as in its column, so we can hope that the same property (almost) holds for a "typical reachable state", although we know that $\eta((\bigotimes_{k=0}^{K-1} \mathbf{A}^k)_{\hat{\mathcal{S}},\mathcal{S}}) \geq \eta((\bigotimes_{k=0}^{K-1} \mathbf{A}^k)_{\mathcal{S},\hat{\mathcal{S}}}) = \eta((\bigotimes_{k=0}^{K-1} \mathbf{A}^k)_{\mathcal{S},\mathcal{S}})$.

Another area of potential improvement is the $\log n$ overhead required to find the index of a state in *ByRows* or *ByColumns*. This can be reduced by observing that the cost to search for a state can be amortized. For example, consider the multiplication procedure *BtrByRows* shown in Fig. 3. Given a pointer I_{k-1} to a "reachable substate" $(\mathbf{i}_0, \ldots \mathbf{i}_{k-1})$, $\rho_k(I_{k-1})$ is the set of local states $\mathbf{i}_k$ such that $(\mathbf{i}_0, \ldots, \mathbf{i}_k)$ is a reachable substate as well (a substate of length k is reachable if it represents the first k components of a state $\mathbf{i} \in \mathcal{S}$). With the appropriate data structure, we can find this set very efficiently. Also, given a pointer J_{k-1} to a reachable substate $(\mathbf{j}_0, \ldots \mathbf{j}_{k-1})$, $\Psi_k(J_{K-1}, \mathbf{j}_k)$ is the pointer to the substate $(\mathbf{j}_0, \ldots \mathbf{j}_k)$, if it is reachable, or the value "null" otherwise. With the same data structure, we can compute this function in time $O(\log \rho_k(J_{k-1})) \leq O(\log n_k)$.

Hence, the overall complexity of *BtrByRows* is $O(\eta((\bigotimes_{k=0}^{K-1} \mathbf{A}^k)_{\mathcal{S},\mathcal{S}}) \cdot \log n_{K-1})$, which is much closer to the "ideal" complexity $O(\eta((\bigotimes_{k=0}^{K-1} \mathbf{A}^k)_{\mathcal{S},\mathcal{S}}))$, even for small values of K, provided $\rho_{K-1}(I_{K-2})$ is neither "too small" nor "too large". Similar (although not as effective) improvements can be applied to a multiplication by columns as well.

BtrByRows(in: $\mathbf{x}$, $\mathbf{A}^0, \ldots, \mathbf{A}^{K-1}$; inout: $\mathbf{y}$)

```
1.  for each i_0 ∈ ρ_0(null)
2.      I_0 ← Ψ_0(null, i_0);
3.      for each j_0 s.t. A^0_{i_0,j_0} > 0
4.          J_0 ← Ψ_0(null, j_0);
5.          a_0 ← A^0_{i_0 j_0};
6.          for each i_1 ∈ ρ_1(I_0)
7.              I_1 ← Ψ_1(I_0, i_1);
8.              for each j_1 s.t. A^1_{i_1,j_1} > 0
9.                  J_1 ← Ψ_1(J_0, j_1);
10.                 a_1 ← a_0 · A^1_{i_1,j_1};
    ...
11.                 for each i_{K-1} ∈ ρ_{K-1}(I_{K-2})
12.                     I_{K-1} ← Ψ_{K-1}(I_{K-2}, i_{K-1});
13.                     for each j_{K-1} s.t. A^{K-1}_{i_{K-1},j_{K-1}} > 0
14.                         J_{K-1} ← Ψ_{K-1}(J_{K-2}, j_{K-1});
15.                         a_{K-1} ← a_{K-2} · A^{K-1}_{i_{K-1} j_{K-1}};
16.                         y_{J_{K-1}} ← y_{J_{K-1}} + x_{I_{K-1}} · a_{K-1};
```

Figure 3: An improved multiplication by rows.

4 Modeling manufacturing systems

Petri nets have been used to model manufacturing systems because they can represent logical behaviors such as synchronizations, fork-and-joins, contentions for resources, decisions, iterations, and sequentializations, but it is easy to envision a domain-specific language or formalism which would simplify the specification process even more.

We envision a modeling approach where each station in the system is described by a submodel with clearly defined interfaces (transitions that can be merged, "synchronized"). Different stations, or multiple instances of the same station, are then combined (by identifying sets of merged transitions), resulting in increasingly larger models.

The compositional approach based on Kronecker algebra is then ideally suited to solve these models, since the designer implicitly specifies the submodels, and no explicit "decomposition" step of a single large model is required [3].

We report early results on an example of a kanban system, where the compositional approach has been used to connect four copies ($k = 0, \ldots, 3$) of the model of a single station, shown in Fig. 4, so that $\{t_{out_0}, t_{in_1}, t_{in_2}\}$ are merged into a single transition, and so are $\{t_{out_1}, t_{out_2}, t_{in_3}\}$ [2].

Table 1 shows the size of the iteration vectors n, of the potential state space $\hat{n}$, and of the matrix that would have been required, η, had we solved the model with conventional methods. The solution times for the

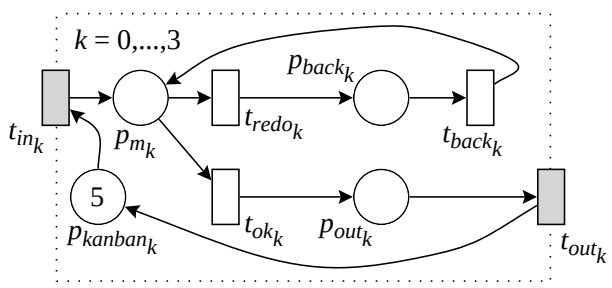

Figure 4: The GSPN for a single kanban station.

Case	n	$\hat{n}$	η	time
A	2,546,432	9,834,496	24,460,016	22,215
B	2,546,432	2,546,432	24,460,016	6,054
C	1,270,962	9,834,496	12,795,046	4,111
D	2,040,442	2,546,432	12,795,046	3,562

Table 1: Computational and storage requirements.

numerical methods are reported as well, in second. In cases A and B, the synchronized transitions are timed, while, in C and D, they are immediate, thus requiring to preserve some of the vanishing states. Furthermore the difference between A and B (and between C and D) is that we consider each kanban as a separate submodel in the former, ($K = 4$), while we group kanban 1 and 2 into into a single submodel in the latter ($K = 3$). Unreachable states arise due to the synchronizations, which ensure that p_{kanban_1} and p_{kanban_2} always contain the same number of tokens. Only cases B and D capture this restriction, hence, for B, $\hat{\mathcal{S}} = \mathcal{S}$, while, for D, $\mathcal{S}$ is still smaller than $\hat{\mathcal{S}}$, because the immediate synchronizations can prevent the occurrence of other timed events in the GSPN. The solution times of B and D are indeed smaller, even if they ultimately solve the same linear systems as their counterparts, A and B.

For even larger models, an approximate solution will be required, since even the storage of a single vector $\boldsymbol{\pi}$ of size n may become prohibitive. The usual approach is then to analyze separate subsystems with exact methods, and iterate among them in a fixed-point fashion. The ability of carrying on the exact solution of larger model will nevertheless be useful, since the approximation is generally better when iterating over a small number of large subsystems, rather than a large number of small subsystems [3].

5 Conclusion and acknowledgments

Compositional approaches based on Kronecker operators have a clear edge over standard approaches requiring the explicit storage of the transition rate matrix, from a memory standpoint. We believe that, with the improvements shown here and others that are no doubt possible, their actual solution times will be only mildly worse for models whose transition rate matrix would barely fit in main memory. For even larger models, the effect of virtual memory with traditional approaches will more than offset the higher overhead for the management of the complex data structures needed by compositional approaches. Thus, we trust that compositional approaches will become commonplace for the solution of truly enormous models.

Some of the results presented were jointly developed with Marco Tilgner (Tokyo Institute of Technology, Japan), Susanna Donatelli (Università di Torino, Italy), and Peter Buchholz and Peter Kemper (Universität Dortmund, Germany).

References

[1] P. Buchholz. Numerical solution methods based on structured descriptions of Markovian models. In G. Balbo and G. Serazzi, editors, *Computer performance evaluation*, pages 251–267. Elsevier Science Publishers B.V. (North-Holland), 1991.

[2] G. Ciardo and M. Tilgner. On the use of Kronecker operators for the solution of generalized stochastic Petri nets. *J. ACM*. Submitted.

[3] G. Ciardo and K. S. Trivedi. A decomposition approach for stochastic reward net models. *Perf. Eval.*, 18(1):37–59, 1993.

[4] S. Donatelli. Superposed generalized stochastic Petri nets: definition and efficient solution. In R. Valette, editor, *Application and Theory of Petri Nets 1994, Lecture Notes in Computer Science 815 (Proc. 15th Int. Conf. on Applications and Theory of Petri Nets, Zaragoza, Spain)*, pages 258–277. Springer-Verlag, June 1994.

[5] P. Kemper. Numerical analysis of superposed GSPNs. In *Proc. Int. Workshop on Petri Nets and Performance Models (PNPM'95)*, pages 52–61, Durham, NC, Oct. 1995. IEEE Comp. Soc. Press.

[6] B. Plateau and K. Atif. Stochastic Automata Network for modeling parallel systems. *IEEE Trans. Softw. Eng.*, 17(10):1093–1108, Oct. 1991.