IN-65
089760

NASA/CR-97-206267
ICASE Report No. 97-67

ICASE 25th ANNIVERSARY

Parametric State Space Structuring

Gianfranco Ciardo
College of William and Mary

Marco Tilgner
Tokyo Institute of Technology

Institute for Computer Applications in Science and Engineering
NASA Langley Research Center
Hampton, VA

Operated by Universities Space Research Association

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23681-2199

Prepared for Langley Research Center
under Contract NAS1-19480

December 1997

PARAMETRIC STATE SPACE STRUCTURING

GIANFRANCO CIARDO* AND MARCO TILGNER†

Abstract. Structured approaches based on Kronecker operators for the description and solution of the infinitesimal generator of a continuous-time Markov chains are receiving increasing interest. However, their main advantage, a substantial reduction in the memory requirements during the numerical solution, comes at a price. Methods based on the "potential state space" allocate a probability vector that might be much larger than actually needed. Methods based on the "actual state space", instead, have an additional logarithmic overhead. We present an approach that realizes the advantages of both methods with none of their disadvantages, by partitioning the local state spaces of each submodel. We apply our results to a model of software rendezvous, and show how they reduce memory requirements while, at the same time, improving the efficiency of the computation.

Key words. Kronecker algebra, Markov chains, structured state space.

Subject classification. Computer Science

1. Introduction. Markov chains can be used to model complex systems exhibiting stochastic behavior, but their numerical solution is limited by the high memory requirements. For continuous-time Markov chains (CTMCs) in particular, three large data structures need to be stored in a conventional solution approach: the state space $\mathcal{T}$, the transition rate matrix $\mathbf{R}$, and the desired solution vector (e.g., the steady-state probabilities $\boldsymbol{\pi}$). Of these, $\mathbf{R}$ is the largest even when using sparse storage, since it contains as many nonzero entries as the number of possible state-to-state transitions.

Thus, much work has been performed in the last decade on techniques to store $\mathbf{R}$ implicitly using Kronecker operators, starting with the "synchronized" stochastic automata of Plateau [15, 16, 19]. Buchholz [1] used a similar idea for Markovian closed (asynchronous) queueing networks, and Takahashi [22] used it for open queueing networks with communication blocking. Donatelli [10, 11] adapted the approach to generalized stochastic Petri nets (GSPNs), introducing the "superposed GSPNs (SGSPNs)", further extended by Kemper [12].

In these approaches, K submodels are "composed" through some rules. For example, in the SGSPNs, individual submodels having an underlying ergodic CTMC are composed by merging transitions in two or more submodels, so that they conceptually fire at the same time. The state space $\mathcal{T}$ of the overall model is a (possibly proper) subset of the cross-product $\hat{\mathcal{T}}$ of the state spaces of the individual submodels. Analogously, the matrix $\mathbf{R}$ is a submatrix of a matrix $\hat{\mathbf{R}}$ obtained by combining a set of small matrices that describe the effect of an event on an individual submodel, using Kronecker products and sums.

The possibility that the "potential" state space $\hat{\mathcal{T}}$ might contain states not in the "actual" state space $\mathcal{T}$ has been addressed in two fundamentally different ways:

- One can ignore the distinction between $\hat{\mathcal{T}}$ and $\mathcal{T}$, and write algorithms that use a vector $\hat{\boldsymbol{\pi}}$ of size

*Department of Computer Science, College of William and Mary, Williamsburg, VA 23187-8795, USA. (email: *ciardo@cs.wm.edu*). This research was partially supported by the National Aeronautics and Space Administration under NASA Contract No. NAS1-19480 while this author was in residence at the Institute for Computer Applications in Science and Engineering (ICASE), NASA Langley Research Center, Hampton, VA 23681.

†Department of Mathematical and Computing Sciences, Tokyo Institute of Technology, 2-12-1 Ookayama, Meguro-ku, Tokyo 152, Japan. (email: *marco@is.titech.ac.jp*).

$|\hat{\mathcal{T}}|$ instead of a vector $\boldsymbol{\pi}$ of size $\mathcal{T}$, and iterate on $\hat{\mathbf{R}}$ [15, 16, 19, 1, 17, 21]. This approach is simpler and does not require to store the state space $\mathcal{T}$, but it might unnecessarily fail for lack of memory when $|\hat{\mathcal{T}}| \gg |\mathcal{T}|$.

- If a vector $\boldsymbol{\pi}$ of size $|\mathcal{T}|$ is used instead, the iterations of the numerical methods must be restricted to operate on the correct submatrix $\mathbf{R}$ of $\hat{\mathbf{R}}$. This can be done by explicitly storing the state space $\mathcal{T}$, and then using the entry $\hat{\mathbf{R}}_{i,j}$ computed through Kronecker operations, if, and only if, both i and j are reachable states in $\mathcal{T}$ [12, 24, 9]. This involves searching for states in the data structure storing $\mathcal{T}$, resulting in an additional logarithmic overhead.

In this contribution, we explore a third possibility, which in the best case achieves the performance of the simpler methods based on $\hat{\mathcal{T}}$ while using even less memory than those based on $\mathcal{T}$. The approach uses a partition of each local state space in such a way that either $\mathcal{T}$ or a superset $\tilde{\mathcal{T}}$ of it (hopefully much smaller than $\hat{\mathcal{T}}$) can be encoded in a negligible amount of space. Furthermore, the question "is this state reachable" can be answered in $O(K)$ time, that is, by simply looking at its components, without having to perform a logarithmic-time search.

While finding the "best partition" that will minimize the memory requirements is unfeasible in general, we show how the presence of invariants in the model can be used to guide us toward a "good partition" that can be used by our algorithm.

The resulting block-partitioning of the matrix $\mathbf{R}$ is analogous to that introduced by Buchholz [2, 4] for the solution of "asynchronous systems", essentially restricted GSPN models composed of submodels where the interactions are not due to merging transitions but to a transition having input(s) in one submodel and output(s) in another. Also related to our work is a recent contribution by Campos, Silva, and Donatelli [5] on "stochastic deterministically synchronized sequential processes".

Our approach, however, is not restricted to a particular pattern of interaction between submodels, and can cope with "imperfect" partitioning, which might result in the most efficient approach in practice, as we show in our example.

The paper is structured as follows. Section 2 describes the notation used and recalls the definition of Kronecker product and sum. Section 3 defines high-level models and their underlying CTMCs. Then, Section 4 describes how both the state space and the transition rate matrix can be described in a structured way from information "local" to each submodel, and recalls the main ideas of the potential and actual state-space-based solution approaches. Sections 5 and 6 introduce our main idea, the partition of local state spaces, and discuss how to find a good partition in practice. The effect of applying the partition to $\mathbf{R}$ is discussed in Section 7, while the resulting equations for the solution of the CTMC are derived in Section 8. Finally, Section 9 uses the new technique to study a software tasking system and details the timing and memory requirements under alternative solution approaches.

2. Notation.

Except for the sets of natural numbers, $\mathbb{N} = \{0, 1, 2 \ldots\}$, and real numbers, $\mathbb{R}$, all sets are denoted by upper-case calligraphic letters (e.g., $\mathcal{X}$); (row) vectors and matrices are denoted by lower- and upper-case bold letters, respectively (e.g., $\mathbf{x}$, $\mathbf{A}$); their entries are denoted by subscripts (e.g., $\mathbf{x}_i$, $\mathbf{A}_{i,j}$); a set of indices can be used instead of a single index, for example, $\mathbf{A}_{\mathcal{X},\mathcal{Y}}$ denotes the submatrix of $\mathbf{A}$ corresponding to set of rows $\mathcal{X}$ and the set of columns $\mathcal{Y}$. We also index families of like-quantities with subscripts or superscripts (e.g., x_i or x^i) and use a shorthand "range" notation to indicate sequences of them (e.g., $x_{[1,n]} = x_1, \ldots, x_n$)

$\eta(\mathbf{A})$ denotes the number of nonzeros in matrix $\mathbf{A}$. $\mathbf{0}_{n\times m}$ and $\mathbf{1}_{n\times m}$ denote matrices with n rows and m columns, having all entries equal 0 or 1, respectively, while $\mathbf{I}_n$ denotes the identity matrix of size $n \times n$;

the dimensions of these matrices may be omitted when they are clear from the context. Given a vector $\mathbf{x}$, $diag(\mathbf{x})$ is a square matrix having vector $\mathbf{x}$ on the diagonal and zero elsewhere.

Given a $n \times n$ matrix $\mathbf{A}$, $rwsm(\mathbf{A}) = diag(\mathbf{A} \cdot \mathbf{1}_{n\times 1})$ is a $n \times n$ matrix having the diagonal equal to the sums of the entries on each row of $\mathbf{A}$, and zero elsewhere.

We recall the definition of the Kronecker product $\mathbf{A} = \bigotimes_{k=1}^{K} \mathbf{A}^k$ of K rectangular matrices $\mathbf{A}^k \in \mathbb{R}^{r_k \times c_k}$, using the convention that the rows and columns of both $\mathbf{A}$ and the matrices $\mathbf{A}^k$ are indexed starting at 0. Let $n_l^u = \prod_{k=l}^{u} n_k$. If we assume a mixed-base numbering scheme, the tuple $i_{[1,K]}$ corresponds to the row index $(\ldots((i_1)r_2 + i_2)r_3 \cdots)r_K + i_K = \sum_{k=1}^{K} i_k \cdot r_{k+1}^K$, and vice versa; the interpretation of a column index $j_{[1,K]}$ is analogous, except that c_k must be used instead of r_k. Then, the generic element of $\mathbf{A} \in \mathbb{R}^{r_1^K \times c_1^K}$ is given as

$$\mathbf{A}_{i,j} = \mathbf{A}_{i_{[1,K]},j_{[1,K]}} = \mathbf{A}^1_{i_1,j_1} \cdot \mathbf{A}^2_{i_2,j_2} \cdots \mathbf{A}^K_{i_K,j_K}. \tag{2.1}$$

The Kronecker sum $\bigoplus_{k=1}^{K} \mathbf{A}^k$ is defined, only for square matrices $\mathbf{A}^k \in \mathbb{R}^{n_k \times n_k}$, in terms of Kronecker products, as

$$\sum_{k=1}^{K} \mathbf{I}_{n_1} \otimes \cdots \otimes \mathbf{I}_{n_{k-1}} \otimes \mathbf{A}^k \otimes \mathbf{I}_{n_{k+1}} \otimes \cdots \otimes \mathbf{I}_{n_K}.$$

We are interested in algorithms that exploit sparsity. For the Kronecker product, the number of nonzeros is simply $\eta(\bigotimes_{k=1}^{K} \mathbf{A}^k) = \prod_{k=1}^{K} \eta(\mathbf{A}^k)$.

Table 2.1 summarizes the symbols used in the paper.

3. Description and solution of a Markov model. We assume that the Markov model is expressed in a structured high-level formalism as a collection of K submodels $m_{[1,K]}$. These define:

- A set of potential states, $\hat{\mathcal{T}} = \times_{k=1}^{K} \hat{\mathcal{T}}^k$, where $\hat{\mathcal{T}}^k = \{0, \ldots, n_k - 1\}$ is the set of n_k potential local states for m_k.
- A set of events, $\mathcal{E}$.
- An initial state, $i^0_{[1,K]} \in \hat{\mathcal{T}}$.
- Rules to define whether an event $e \in \mathcal{E}$ is active in a state $i_{[1,K]} \in \hat{\mathcal{T}}$, $act(e, i_{[1,K]}) \in \{True, False\}$,
- its rate of occurrence when active, $rt(e, i_{[1,K]}) > 0$,
- the state obtained when e occurs in state $i_{[1,K]}$, $new(e, i_{[1,K]}) \in \hat{\mathcal{T}}$.

From these, we can build:

- A set of reachable states having an exponentially distributed holding time, or state-space, $\mathcal{T}$, and the sets of local state-spaces $\mathcal{T}^k$, simply the projection of $\mathcal{T}$ on its k-th component. Without loss of generality, we assume from now on that $\mathcal{T}^k = \hat{\mathcal{T}}^k$.
- A transition rate matrix, $\mathbf{R} \in \mathbb{R}^{|\mathcal{T}|\times|\mathcal{T}|}$, describing the transition rates between reachable states.
- An initial probability vector, $\boldsymbol{\pi}(0) \in \mathbb{R}^{|\mathcal{T}|}$.

$\mathcal{T}$ can be generated using a state-space exploration algorithm, such as *BuildRS*, shown in Fig. 3.1, which terminates iff $\mathcal{T}$ is finite. $\mathcal{T}$ and $\mathcal{U}$ are the sets of states explored so far, or found but not yet explored, respectively. If $\mathcal{U}$ is managed as a FIFO linked list, *BuildRS* performs a breadth-first search of the graph implicitly defined by the model. The matrix $\mathbf{R}$ can be generated at the same time, or in a second pass, once $|\mathcal{T}|$ and $\eta(\mathbf{R})$ are known, so that an efficient sparse row-wise or column-wise format can be used [14].

The solution algorithms we discuss index the states according to "lexicographic order", $\Psi : \mathcal{T} \rightarrow \{0, \ldots, |\mathcal{T}| - 1\}$, such that $\Psi(i_{[1,K]}) > \Psi(j_{[1,K]})$ iff $i_{[1,K]}$ follows $j_{[1,K]}$ in a lexicographic comparison. This

TABLE 2.1

Symbols used in the paper

Symbol	Meaning
K	Number of submodels
m_k	k-th submodel
n_k	Size of potential local state space for m_k
n_l^u	$\prod_{k=l}^{u} n_k$
$\hat{\mathcal{T}}, \mathcal{T}$	Potential, actual state space
$\hat{\mathcal{T}}^k, \mathcal{T}^k$	Potential, actual local state space for m_k
$\Psi(i_{[1,K]})$	Lexicographic position of $i_{[1,K]}$
$\mathcal{E}, \mathcal{E}^k, \mathcal{E}^*$	Events: overall, local to m_k, synchronizing
$\hat{\mathbf{R}}, \mathbf{R}$	Transition rate matrix
$\mathbf{R}^k$	Local transition rate matrix for m_k
$\mathbf{W}^{k,l}$	Weight matrix for synchr. event e_l on m_k
$\hat{\boldsymbol{\pi}}, \boldsymbol{\pi}$	Steady state probability vector
$\hat{\mathbf{h}}, \mathbf{h}$	Expected holding time vector
P_k	Number of classes in the partition of $\mathcal{T}^k$
$\mathcal{P}^k$	$\{0, \ldots, P_k - 1\}$, class indices
$\mathcal{T}^{k:p}$	p-th class in the partition, $p \in \mathcal{P}^k$
$n_{k:p}$	$\lvert\mathcal{T}^{k:p}\rvert$, size of the p-th class
$\mathcal{R}$	Relation defined on the sets $\mathcal{P}^k$
$\mathbf{B}$	Blocks defining $\mathbf{R}$

```
$BuildRS$(in: $\mathcal{E}, act, new$; out: $\mathcal{T}$);
    1. $\mathcal{T}$      $\emptyset$;
    2. $\mathcal{U}$      $\{i^0_{[1,K]}\}$;
    3. while $\mathcal{U} \neq \emptyset$
    4.    "remove the first state $i_{[1,K]}$ from $\mathcal{U}$";
    5.    $\mathcal{T}$      $\mathcal{T} \cup \{i_{[1,K]}\}$;
    6.    for each $e_l \in \mathcal{E}$ s.t. $act(e_l, i_{[1,K]})$
    7.       $j_{[1,K]}$      $new(e_l, i_{[1,K]})$;
    8.       if $j_{[1,K]} \notin \mathcal{T} \cup \mathcal{U}$ then
    9.          $\mathcal{U}$      $\mathcal{U} \cup \{j_{[1,K]}\}$;
```

FIG. 3.1. *Algorithm BuildRS*

assumes that states (or their encodings) are somehow comparable. Furthermore, we extend Ψ so that, $\forall i_{[1,K]} \in \hat{\mathcal{T}} \setminus \mathcal{T}$, $\Psi(i_{[1,K]}) = \text{null}$.

We focus on the computation of the steady-state probability (row) vector $\boldsymbol{\pi} \in I\!R^{|\mathcal{T}|}$ satisfying

$$\boldsymbol{\pi} \cdot (\mathbf{R} - rwsm(\mathbf{R})) = \mathbf{0}_{1\times|\mathcal{T}|} \qquad \text{and} \qquad \boldsymbol{\pi} \cdot \mathbf{1}_{|\mathcal{T}|\times 1} = 1, \tag{3.1}$$

but the techniques we discuss are equally applicable to the computation of transient and cumulative measures [6]. We note that the indices of $\boldsymbol{\pi}$ and $\mathbf{R}$ correspond to the reachable states through the mapping Ψ, that is, the steady-state probability of state $i_{[1,K]} \in \mathcal{T}$ is stored in position $\Psi(i_{[1,K]})$ of $\boldsymbol{\pi}$.

There are two difficulties in solving the system of linear equations (3.1). First, models of interest can describe CTMCs of enormous size, preventing us from using direct methods such as Gaussian elimination, which cause excessive fill-in. We are then limited to iterative methods using sparse storage schemes for $\mathbf{R}$, but even these methods are memory-bound when applied to realistic examples. Furthermore, the convergence rate of iterative methods can be quite poor when applied to stiff models, such as those arising in reliability analysis.

In the following section, we recall results from [8, 9, 3] showing how the state-space $\mathcal{T}$ and the matrix $\mathbf{R}$ can be represented in compact form using Kronecker operators, thus reducing the storage requirements, but possibly at the cost of execution efficiency. Then, we show how, by an appropriate partition of the local state-spaces, we can achieve better performance, while further reducing the storage requirements.

4. Structured description of $\mathcal{T}$ and R. In [8], we showed how $\mathcal{T}$ can be stored in an efficient dynamic multi-tree data structure during the execution of *BuildRS*. After $\mathcal{T}$ is known, this data structure can be further compressed by using arrays instead of dynamically balanced search trees, as shown in Fig. 4.1.

Using the data structure in Fig. 4.1 to compute $\Psi(i_{[1,K]})$ is straightforward:

1. Use binary search to find i_1 in the array for m_1 and follow the pointer to the array for m_2.
2. Search i_2 in the array for m_2 (only the grey portion of size $\leq n_2$ between the pointed position and the next pointed position must be considered). If found, follow the pointer to the array for m_3.
3. Continue to follow the pointers until either a local state i_k is not found (implying that $i_{[1,K]} \notin \mathcal{T}$), or until i_K is found (implying that $i_{[1,K]} \in \mathcal{T}$ and that its lexicographic position $\Psi(i_{[1,K]})$ is the same as the position where i_K was found in the array for m_K).

Assuming that the portions of the arrays searched at each level are of comparable size, the overall complexity to compute $\Psi(i_{[1,K]})$ is $O(\log|\mathcal{T}|)$, the same as if $\mathcal{T}$ were stored in a single search tree or in a contiguous array. However, there are several advantages:

- The memory requirements to store $\mathcal{T}$ using this data structure can be as low as $O(\lceil \log n_K \rceil \cdot \mathcal{T})$ bits, as long as, for any sub-state $i_{[1,K-1]}$, the sets of local states $\{i_K \in \mathcal{T}^K : i_{[1,K]} \in \mathcal{T}\}$ are either empty or reasonably large. In Fig. 4.1, this requirement implies that the last array, for m_K, should be substantially larger than the array for m_{K-1}, and so on. The straightforward data structure requires instead $O(\sum_{k=1}^{K} \lceil \log n_k \rceil \cdot \mathcal{T})$ bits.
- Only local states are compared at each step, not global states as in the straightforward algorithm.
- It is often possible to realize that $i_{[1,K]} \notin \mathcal{T}$ without having to consider all its components (on the other hand, it is impossible to determine that $i_{[1,K]} \in \mathcal{T}$ before examining the array for m_K).
- After searching for $i_{[1,K]}$, a search for a state with the same first k components can reuse the work already performed to find the path corresponding to $i_{[1,k]}$, and start directly at the array for m_{k+1}. This is the key to lowering the complexity of the vector-matrix multiplications presented in [3].

In practice, our implementation automatically uses 8-, 16- or 32-bit indices for the local states of m_k, depending on the value of n_k. For most models, $n_K \leq 2^8$ or 2^{16}, hence we can store $\mathcal{T}$ in little over $|\mathcal{T}|$ or $2|\mathcal{T}|$ bytes using the data structure of Fig. 4.1.

Following [9, 3], we assume that the activation, rate, and effect of an event can be determined in a "distributed fashion". In other words, this implies that the effects of the various submodels on the event are independent of each other. Formally, given an event $e_l \in \mathcal{E}$:

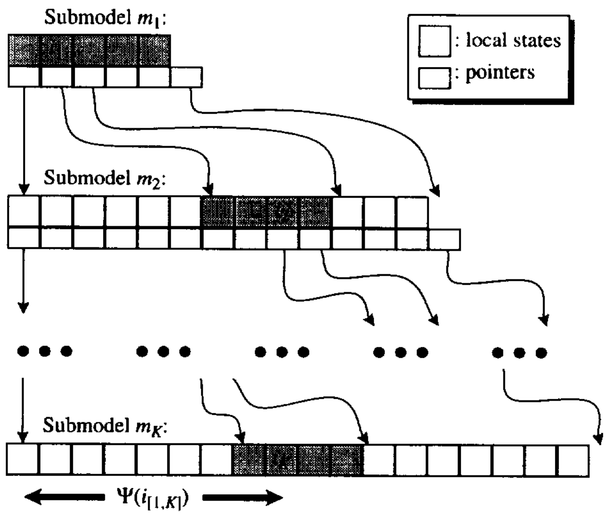

FIG. 4.1. *Efficient storage of $\mathcal{T}$.*

- e_l is active in $i_{[1,K]}$ iff it is "locally active" in each local state:

(4.1) $$act(e_l, i_{[1,K]}) = act^{1,l}(i_1) \wedge \cdots \wedge act^{K,l}(i_K)$$

  where $act^{k,l} : \mathcal{T}^k \rightarrow \{True, False\}$ is derived from the model.
- The effect of the occurrence of e_l on a local state does not depend on the other local states:

(4.2) $$new(e_l, i_{[1,K]}) = (new^{1,l}(i_1), \ldots, new^{K,l}(i_K))$$

  where $new^{k,l} : \mathcal{T}^k \rightarrow \mathcal{T}^k$ is given by the model.
- Finally, the rate of an active event e_l in state $i_{[1,K]}$ can be expressed as

(4.3) $$rt(e_l, i_{[1,K]}) = \lambda_l \cdot wgt^{1,l}(i_1) \cdots wgt^{K,l}(i_K)$$

  where $\lambda_l \in \mathbb{R}^+$ is a constant "reference rate" and $wgt^{k,l} : \mathcal{T}^k \rightarrow \mathbb{R}^+$ are (dimensionless) scaling weights, also given by the model.

If the activation, effect, or rate of an event e_l are not dependent on a local state i_k, we let $act^{k,l} \equiv True$, $new^{k,l} \equiv \mathbf{I}$, the identity function, and $wgt^{k,l} \equiv 1$, respectively.

We can then partition the set of events $\mathcal{E}$. An event e_l is local to m_k, we write $e_l \in \mathcal{E}^k$, if its activation, rate, and effect are determined by, and limited to, the local state i_k of m_k alone:

$$\mathcal{E}^k \overset{\text{def}}{=} \Big\{ e_l \in \mathcal{E} : \forall k' \neq k,\ act^{k',l} \equiv True \wedge\ new^{k',l} \equiv \mathbf{I}\ \wedge\ wgt^{k',l} \equiv 1 \Big\}.$$

The events in

$$\mathcal{E}^* = \mathcal{E} \setminus \bigcup_{k=1}^{K} \mathcal{E}^k$$

are instead said to be synchronizing. For notational convenience, these are numbered first: $e_l \in \mathcal{E}^*$ iff $1 \leq l \leq L = |\mathcal{E}^*|$.

It has been shown in [9] (but see also [15, 16, 1, 11] for more restrictive statements) that, when the conditions specified in (4.1), (4.2), and (4.3) hold, $\mathbf{R}$ can be expressed as

$$\mathbf{R} = \hat{\mathbf{R}}_{\mathcal{T},\mathcal{T}} = \left(\bigoplus_{k=1}^{K} \mathbf{R}^k + \sum_{l=1}^{L} \lambda_l \cdot \bigotimes_{k=1}^{K} \mathbf{W}^{k,l} \right)_{\mathcal{T},\mathcal{T}} , \tag{4.4}$$

where

- $\hat{\mathbf{R}}$ is a $\hat{\mathcal{T}} \times \hat{\mathcal{T}}$ matrix that can be described as the sum of Kronecker products and sums of much smaller matrices, and has the fundamental property that its entries in the rows and columns corresponding to states in $\mathcal{T}$ coincide in value with those of $\mathbf{R}$.
- $\mathbf{W}^{k,l}$ is a $n_k \times n_k$ "weight" matrix defined as

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

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

$$\mathbf{R}^k = \sum_{e_l \in \mathcal{E}^k} \lambda_l \cdot \mathbf{W}^{k,l}.$$

This description of $\mathbf{R}$ can then be used in iterative methods such as Jacobi, Gauss-Seidel, or their relaxation versions [3], avoiding the large storage requirements for $\mathbf{R}$ entirely. Only local matrices such as $\mathbf{W}^{k,l}$ and $\mathbf{R}^k$ need to be stored, and their memory requirements are small even for reasonably large K and L. The only remaining memory constraint is the storage of the iteration vector(s). However, these approaches based on Kronecker operators still suffer from important limitations, which we now address.

4.1. Implementations based on $\hat{\mathcal{T}}$. In the first implementations that appeared in the literature [1, 15, 16, 18], a vector $\hat{\pi} \in I\!R^{n_1^K}$ is used to store π. In it, only the entries corresponding to states $i_{[1,K]} \in \mathcal{T}$ are actually used, while the remaining entries, corresponding to potential but not reachable states, are always zero.

All Kronecker operations are performed ignoring the distinction between reachable and potential states (testing for zero can still be used to avoid some, but not all, useless computations). The main advantage of this approach is that, given a state $i_{[1,K]}$, the position of its corresponding entry in $\hat{\pi}$ can be computed in $O(K)$ operations, as $\sum_{k=1}^{K} i_k \cdot n_{k+1}^K$.

In addition, exploiting the structure of the matrix resulting from a Kronecker product, the vector-matrix product $\hat{\mathbf{x}} \cdot (\bigotimes_{k=1}^{K} \mathbf{A}^k)$ can be computed in $O(n_1^K \cdot \sum_{k=1}^{K} \eta(\mathbf{A}^k)/n_k)$ operations, instead of $O(\prod_{k=1}^{K} \eta(\mathbf{A}^k))$. This result can be obtained by extending the Plateau-Stewart Theorem 9.1 of [20] to the case where the matrices $\mathbf{A}^k$ are stored with an appropriate sparse format.

Unfortunately, the practical impact of this elegant result might not be as great as one could hope because the matrices involved are often "ultrasparse", with $\eta(\mathbf{A}^k)$ at most slightly larger than n_k. Assuming ϵ nonzeros per row, the complexity is then $O(n_1^K \cdot K\epsilon)$ versus $O(n_1^K \cdot \epsilon^K)$ for the straightforward multiplication, which is then a better choice whenever $\epsilon < e^{\frac{\log K}{K-1}}$. For example, when $\epsilon = 1$, application of the Plateau-Stewart Theorem results in a complexity $O(n_1^K \cdot K)$, worse than the $O(n_1^K)$ complexity of the straightforward multiplication (of course, the memory savings might still justify using a Kronecker approach).

4.2. Implementations based on $\mathcal{T}$. Alternative implementations that operate directly on the vector π, thus avoiding $O(n_1^K)$ memory requirements, have been recently proposed [9, 3, 12]. This is an important advantage whenever $|\mathcal{T}| \ll |\hat{\mathcal{T}}|$, often the case in practice.

However, these memory savings come with a price. When using $\hat{\mathcal{T}}$, a state $i_{[1,K]}$ can be easily associated with its index in $\hat{\boldsymbol{\pi}}$, while now we must compute $\Psi(i_{[1,K]})$ to obtain the index in $\boldsymbol{\pi}$, using a logarithmic search in $\mathcal{T}$. A simplistic implementation of the vector-matrix product

$$\mathbf{x} \cdot \left.\bigotimes_{k=1}^{K} \mathbf{A}^k\right)_{\mathcal{T},\mathcal{T}}$$

has then complexity

$$O\left(\eta\left.\bigotimes_{k=1}^{K} \mathbf{A}^k\right)_{\mathcal{T},\mathcal{T}} \cdot \log|\mathcal{T}|\right)$$

if performed "by rows", as needed for a Jacobi-style iteration, or

$$O\left(\eta\left.\bigotimes_{k=1}^{K} \mathbf{A}^k\right)_{\hat{\mathcal{T}},\mathcal{T}} \cdot \log|\mathcal{T}|\right)$$

if performed "by columns", as needed for the faster Gauss-Seidel-style iteration (the difference is that spurious entries from states in $\hat{\mathcal{T}} \setminus \mathcal{T}$ to states in $\mathcal{T}$ might be encountered in the latter case).

In [3], the logarithmic factor $\log|\mathcal{T}|$ is reduced to $\log n_K$ with the help of the data structure of Fig. 4.1, but it is not completely eliminated.

5. Partitioning the local state spaces. We now demonstrate how an appropriate partition of the local state-spaces can achieve the advantages of the methods based on either the potential or the actual state space, without many of their drawbacks.

Partition each local state space $\mathcal{T}^k$ into P_k classes

$$\mathcal{T}^k = \bigcup_{p=0}^{P_k-1} \mathcal{T}^{k:p},$$

let $n_{k:p} = |\mathcal{T}^{k:p}|$ be the number of states in class $\mathcal{T}^{k:p}$, and define $\mathcal{P}^k = \{0, \ldots, P_k - 1\}$ to be set of indices corresponding to the classes in the partition. We can then define a relation $\mathcal{R} \subseteq \times_{k=1}^{K} \mathcal{P}^k$ over these sets of indices. Two interesting cases arise, which we denote as perfect and imperfect partition, respectively.

5.1. Perfect partition. Assume that $\mathcal{R}$ can be defined so that

$$p_{[1,K]} \in \mathcal{R} \Rightarrow \mathcal{T} \supseteq \times_{k=1}^{K} \mathcal{T}^{k:p_k} \quad \text{and} \tag{5.1}$$

$$p_{[1,K]} \notin \mathcal{R} \Rightarrow \mathcal{T} \cap \times_{k=1}^{K} \mathcal{T}^{k:p_k} = \emptyset. \tag{5.2}$$

Then, this means that the actual state space $\mathcal{T}$ can be described very compactly as the set of states contained in the cross-products of the classes corresponding to the elements of $\mathcal{R}$:

$$\mathcal{T} = \bigcup_{p_{[1,K]} \in \mathcal{R}} \times_{k=1}^{K} \mathcal{T}^{k:p_k}.$$

Note that, since the P_k classes constitute a partition of $\mathcal{T}^k$, these cross-products constitute a partition of $\mathcal{T}$, that is, given two tuples $p_{[1,K]}$ and $q_{[1,K]}$ differing in at least one component, the cross-products $\times_{k=1}^{K} \mathcal{T}^{k:p_k}$ and $\times_{k=1}^{K} \mathcal{T}^{k:q_k}$ contain disjoint sets of reachable states.

Hence, we can encode the entire state space $\mathcal{T}$ using only a description of the local states for each submodel plus a boolean vector of size $\prod_{k=1}^{K} P_k$, to describe the relation $\mathcal{R}$. We call this a "level-1" vector.

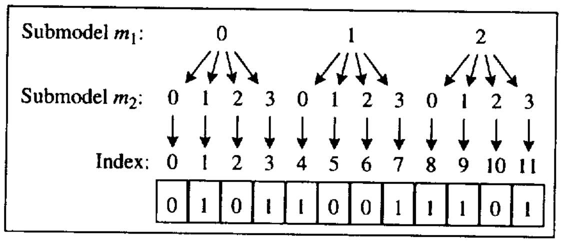
Submodel m_1: 0 1 2
Submodel m_2: 0 1 2 3 0 1 2 3 0 1 2 3
Index: 0 1 2 3 4 5 6 7 8 9 10 11
0 1 0 1 1 0 0 1 1 1 0 1

FIG. 5.1. *An example of perfect partition.*

Of course, $\mathcal{R}$ can always be found, in the worst case by defining a partition of $\mathcal{T}^k$ where each state is in a class by itself, but this results in $\mathcal{R} = \mathcal{T}$ and the level-1 vector has size $|\hat{\mathcal{T}}|$, as the one used by Kemper [12]. On the other hand, when $\hat{\mathcal{T}} = \mathcal{T}$, (5.1) and (5.2) are trivially satisfied by defining a partition of $\mathcal{T}^k$ into a single class: $P_k = 1$ and $\mathcal{T}^k = \mathcal{T}^{k:0}$. In this case, $\mathcal{R} = \{(0, \ldots, 0)\}$.

The case of practical interest is when the actual state space $\mathcal{T}$ is a strict subset of the potential state space $\hat{\mathcal{T}}$, but $1 < P_k \ll n_k$. Then, large savings can be achieved with respect to traditional techniques to store the state space [8] or to Kemper's boolean vector.

As an example, consider a model with two submodels, whose local state spaces can be partitioned as $\mathcal{T}^1 = \mathcal{T}^{1:0} \cup \mathcal{T}^{1:1} \cup \mathcal{T}^{1:2}$ and $\mathcal{T}^2 = \mathcal{T}^{2:0} \cup \mathcal{T}^{2:1} \cup \mathcal{T}^{2:2} \cup \mathcal{T}^{2:3}$, in such a way that, $\forall (i_1, i_2) \in \mathcal{T}^1 \times \mathcal{T}^2$,

$$\begin{aligned}(i_1, i_2) \in \mathcal{T} \Longleftrightarrow & (i_1 \in \mathcal{T}^{1:0} \wedge i_2 \in \mathcal{T}^{2:1} \cup \mathcal{T}^{2:3}) \ \vee \\ & (i_1 \in \mathcal{T}^{1:1} \wedge i_2 \in \mathcal{T}^{2:0} \cup \mathcal{T}^{2:3}) \ \vee \\ & (i_1 \in \mathcal{T}^{1:2} \wedge i_2 \in \mathcal{T}^{2:0} \cup \mathcal{T}^{2:1} \cup \mathcal{T}^{2:3})\end{aligned}$$

Then, the 12-element boolean vector shown in Fig. 5.1 can be used to encode the state space regardless of the sizes of the classes in the partitions (of course, the local state spaces for each submodel still need to be stored explicitly).

To determine whether a given state $i_{[1,K]}$ is reachable, we simply obtain each index p_k of the partition $\mathcal{T}^{k:p_k}$ containing the local state i_k, we compute the mixed-base value of $p_{[1,K]}$,

$$\left. \sum_{k=1}^{K} p_k \cdot \prod_{l=k+1}^{K} P_l \right),$$

and we check whether there is a "1" in the corresponding position of the boolean vector describing $\mathcal{R}$. The complexity of this test is then $O(K)$, independent of the size of $\mathcal{T}$.

5.2. Imperfect partition. At times, the conditions in (5.1) and (5.2) might hold only for excessively fine partitions of the local state spaces, or maybe only in the limiting case $P_k = n_k$, where each state is in a class by itself. To avoid large storage requirements for $\mathcal{R}$, we might relax requirement (5.1), substituting it with

(5.3) $$p_{[1,K]} \in \mathcal{R} \Rightarrow \mathcal{T} \cap \times_{k=1}^{K} \mathcal{T}^{k:p_k} \neq \emptyset$$

which, together with (5.2), simply results in

(5.4) $$p_{[1,K]} \in \mathcal{R} \Longleftrightarrow \mathcal{T} \cap \times_{k=1}^{K} \mathcal{T}^{k:p_k} \neq \emptyset$$

Then, for each $p_{[1,K]} \in \mathcal{R}$, we need to encode which states in $\times_{k=1}^{K} \mathcal{T}^{k:p_k}$ are actually reachable. One way of doing this is to associate a "level-2" boolean vector of size $\prod_{k=1}^{K} n_{k:p_k}$ to each element $p_{[1,K]} \in \mathcal{R}$. This

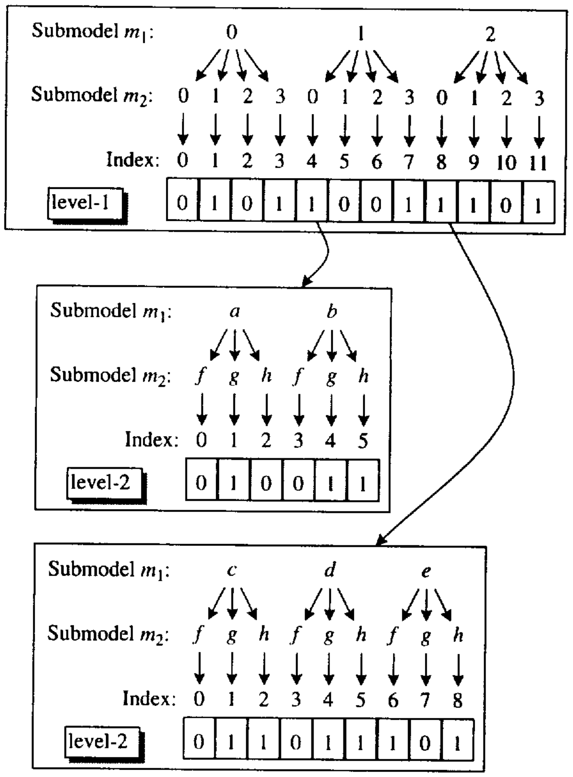

FIG. 5.2. *An example of imperfect partition.*

approach is advantageous with respect to the full boolean vector employed by Kemper [12] provided that the overall memory required by the level-1 and level-2 boolean vectors is substantially less than $|\hat{\mathcal{T}}|$ bits. A condition for this is that the level-1 vector used to encode $\mathcal{R}$ has many "holes", $|\mathcal{R}| \ll \prod_{k=1}^{K} P_k$. However, we must also require that these holes do not correspond only to small classes in the partition.

This is apparent by considering the following extreme case: each local state space $\mathcal{T}^k$ is partitioned into two classes, $\mathcal{T}^{k:0}$ with one state and $\mathcal{T}^{k:1}$ with $n_k - 1$ states, and

$$\mathcal{R} = \{(1,0,\ldots,0),\ldots,(0,\ldots,0,1),(1,\ldots,1)\}.$$

Then, $|\mathcal{R}| = K + 1 \ll \prod_{k=1}^{K} P_k = 2^K$. However, the storage for the boolean vector corresponding to the element $(1,\ldots,1)$ of $\mathcal{R}$ would still require $\prod_{k=1}^{K}(n_k - 1) = O(|\hat{\mathcal{T}}|)$ bits.

Continuing our example, assume now that a "1" in the 12-element boolean vector simply means that some, not necessarily all, of the corresponding global states are reachable. Then, the two-level approach shown in Fig. 5.2 could be used. In the figure, only the level-2 boolean vectors corresponding to the level-1 values (1,0) and (2,0) are shown, assuming that $\mathcal{T}^{1:1} = \{a, b\}$, $\mathcal{T}^{1:2} = \{c, d, e\}$, and $\mathcal{T}^{2:0} = \{f, g, h\}$. For example, the entries in the level-2 boolean vector [010011] pointed by the level-1 value (1,0) tell us that only states (a, g), (b, g), and (b, h) are reachable among those in $\mathcal{T}^{1:1} \times \mathcal{T}^{2:0}$.

5.3. Comparison with alternative approaches.

Two competitive alternatives proposed to store the state space $\mathcal{T}$ are a single boolean vector of size $|\hat{\mathcal{T}}|$ [12] and the multilevel search data structure of Fig. 4.1, requiring little over $|\mathcal{T}|\lceil \log n_K \rceil$ bits [8]. From a memory standpoint, the latter is superior when

$|\hat{\mathcal{T}}|/|\mathcal{T}| > \lceil \log n_K \rceil$. However, it has higher execution overhead, $\log|\mathcal{T}|$ instead of K, when used to determine whether $i_{[1,K]} \in \mathcal{T}$.

In the perfect partition case (assuming non-trivial partitions), the approach we just introduced can use negligible amounts of memory while, in the imperfect partition case, it can have worst-case memory requirements similar to that of Kemper's single boolean vector approach, although one would hope that it would perform much better than that in practice. In either case, the execution complexity to search for a state is only $O(K)$, due to the direct access capabilities of the level-1 and level-2 boolean vectors.

Indeed, one could simply consider our storage approach as a specialized way to compress a boolean vector of size $|\hat{\mathcal{T}}|$ by eliminating entire sets of zero elements at a time. If, for a particular $p_{[1,K]}$, the level-2 boolean vector happens to contain only 1's (if $\mathcal{T} \supseteq \times_{k=1}^{K} \mathcal{T}^{k:p_k}$), we could treat this as a special case and avoid storing it altogether. We ignore this possibility, although it could be exploited in practical implementations. Clearly, the perfect partition case corresponds to the situation where none of the level-2 vector must be allocated explicitly.

We also observe that there is no reason to limit the approach to two levels. The information conveyed by a level-2 boolean vector could be stored by further partitioning the classes of local states it corresponds to, thus introducing a level-3 vector, and so on.

Ultimately, the only limiting factor in this process is the algorithm to decide whether and how to further partition a set of local states. Section 6 addresses this concern in practical situations.

6. Determining a good partition.

We can formulate the problem of finding the best perfect partition as a minimization problem. Given $\mathcal{T}^1, \ldots, \mathcal{T}^K$ and $\mathcal{T} \subseteq \times_{k=1}^{K} \mathcal{T}^k$,

$$\begin{aligned} &\textit{Minimize} && \prod_{k=1}^{K} P_k \\ &\textit{subject to} && \forall k, 1 \le k \le K, \\ & && \mathcal{T}^{k:0}, \ldots, \mathcal{T}^{k:P_k-1} \text{ is a partition of } \mathcal{T}^k, \\ & && \forall p_{[1,K]} \in \times_{k=1}^{K}\{0, \ldots, P_k - 1\}, \\ & && (\mathcal{T} \supseteq \times_{k=1}^{K} \mathcal{T}^{k:p_k}) \vee (\mathcal{T} \cap \times_{k=1}^{K} \mathcal{T}^{k:p_k} = \emptyset). \end{aligned}$$

For an imperfect partition, the quantity to be minimized must include the space for both the level-1 and the level-2 boolean vectors:

$$\begin{aligned} &\textit{Minimize} && \prod_{k=1}^{K} P_k + \sum_{p_{[1,K]} \in \mathcal{R}} \prod_{k=1}^{K} n_{k:p_k} \\ &\textit{subject to} && \forall k, 1 \le k \le K, \\ & && \mathcal{T}^{k:0}, \ldots, \mathcal{T}^{k:P_k-1} \text{ is a partition of } \mathcal{T}^k, \\ & && \mathcal{R} \subseteq \times_{k=1}^{K}\{0, \ldots, P_k - 1\}, \\ & && \forall p_{[1,K]} \in \times_{k=1}^{K}\{0, \ldots, P_k - 1\}, \\ & && p_{[1,K]} \in \mathcal{R} \Longleftrightarrow (\mathcal{T} \cap \times_{k=1}^{K} \mathcal{T}^{k:p_k} \neq \emptyset) \end{aligned} \tag{6.1}$$

The second component of (6.1) is minimized when the number of "0" entries in the level-2 boolean vectors,

$$\left(\sum_{p_{[1,K]} \in \mathcal{R}} \prod_{k=1}^{K} n_{k:p_k} \right) - |\mathcal{T}| \tag{6.2}$$

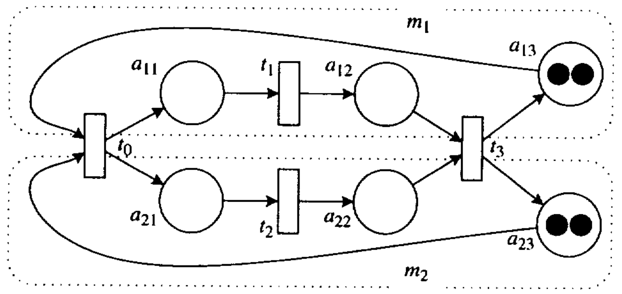

FIG. 6.1. *A simple fork/join model.*

is minimized. Indeed, for a perfect partition, (6.2) is zero.

The search for the optimal partition is then equivalent to an integer programming problem, which is too complex for realistic applications.

We then require a user-defined state-dependent partitioning function

$$part^k : \mathcal{T}^k \rightarrow \mathcal{P}^k$$

for each submodel m_k, assuming that the decomposition of the model into K submodels is defined a priori. In most cases this is naturally given by a modular modeling approach, such as in top-down or bottom-up system design (see also [7] for a definition of a similar partitioning function used for a completely different purpose, the distributed generation of the state space).

Then, the classes of the local state space partition $\mathcal{T}^{k:p}$ are

$$\mathcal{T}^{k:p} = \{i_k \mid p = part^k(i_k)\}$$

and the resulting relation $\mathcal{R}$ is

$$\mathcal{R} = \bigcup_{i_{[1,K]} \in \mathcal{T}} \left\{ \left(part^1(i_1), \ldots, part^K(i_K) \right) \right\}.$$

In other words, we try to find appropriate functions $part^k$ to achieve a (hopefully perfect) partition. For example, in the application presented in Section 9, we will use the total number of "tasks" within a submodel (i.e., its load) as the parameter defining our partition, so that only synchronizations between submodels can result in load changes in the involved submodels. An analogous definition is possible whenever a concept of load can be clearly defined at the submodel level.

For GSPN models having an irreducible underlying Markov chain, this might simply reflect the existence of invariants in the model (sets of places for which a weighted sum of the number of tokens in them is a constant [13]).

For example, consider the simple fork-and-join model of Fig. 6.1. Local states of submodel m_k are expressed as triplets $(\#(a_{k1})\#(a_{k2})\#(a_{k3}))$, where $\#(a)$ indicates the number of tokens in place a. The local state spaces for the two submodels are exactly the same,

$$\mathcal{T}^1 = \mathcal{T}^2 = \{(002), (011), (101), (020), (110), (200)\}.$$

However, not all $n_1 \cdot n_2 = 36$ combinations of local states for m_1 and m_2 are reachable.

In this GSPN, there are four p-invariants:

$$\begin{aligned}
I_{11} &: \#(a_{11}) + \#(a_{12}) + \#(a_{13}) = 2,\\
I_{12} &: \#(a_{11}) + \#(a_{12}) + \#(a_{23}) = 2,\\
I_{21} &: \#(a_{21}) + \#(a_{22}) + \#(a_{13}) = 2,\\
I_{22} &: \#(a_{21}) + \#(a_{22}) + \#(a_{23}) = 2.
\end{aligned}$$

Of these, I_{11} and I_{22} are "local" invariants, they only affect the reachable local states of m_1 and m_2, respectively. The other two "global" invariants I_{12} and I_{21}, though, link the reachable local states of m_1 and m_2.

In this simple case, the invariants tell us exactly which combinations are reachable:

$$\begin{aligned}
\#(a_{23}) = 0 \;&\Rightarrow\; \#(a_{11}) + \#(a_{12}) = 2 \;\Rightarrow\; \#(a_{13}) = 0,\\
\#(a_{23}) = 1 \;&\Rightarrow\; \#(a_{11}) + \#(a_{12}) = 1 \;\Rightarrow\; \#(a_{13}) = 1,\\
\#(a_{23}) = 2 \;&\Rightarrow\; \#(a_{11}) + \#(a_{12}) = 0 \;\Rightarrow\; \#(a_{13}) = 2,
\end{aligned}$$

hence, any global state $(i_1, i_2) \in \mathcal{T}^1 \times \mathcal{T}^2$ of the form $((\#(a_{11}), \#(a_{12}), \#(a_{13})), (\#(a_{21}), \#(a_{22}), \#(a_{23})))$ satisfying $\#(a_{13}) = \#(a_{23})$ is reachable (and no other is):

$$\begin{aligned}
\mathcal{T} = \; \{&((002),(002)),\; ((011),(011)),\; ((011),(101)),\\
&((101),(011)),\; ((101),(101)),\; ((020),(020)),\\
&((020),(110)),\; ((020),(200)),\; ((110),(020)),\\
&((110),(110)),\; ((110),(200)),\; ((200),(020)),\\
&((200),(110)),\; ((200),(200))\;\}.
\end{aligned}$$

Now, if we think of places a_{k3} as "idle" and a_{k1} and a_{k2} as "busy", we can define the load as the number of tokens in the busy places, and define the partitioning functions

$$part^k \equiv \#(a_{k1}) + \#(a_{k2}) = 2 - \#(a_{k3}).$$

This choice partitions the local state spaces $\mathcal{T}^k$ into $P_k = 3$ classes each,

$$\begin{aligned}
\mathcal{T}^{k:0} &= \{\; (002)\; \},\\
\mathcal{T}^{k:1} &= \{\; (011),\; (101)\; \}, \text{ and}\\
\mathcal{T}^{k:2} &= \{\; (020),\; (110),\; (200)\; \},
\end{aligned}$$

Given our choice of partition for the local state spaces, we obtain $\mathcal{R} = \{(0,0), (1,1), (2,2)\}$, resulting in a perfect partition.

We should note that our choice of local partitioning functions is such that $part^k : \mathcal{T}^k \to \mathcal{P}^k$, for each submodel m_k, $1 \le k \le K$, is invariant with respect to the occurrence of any local event:

$$\forall e_l \in \mathcal{E}^k,\; \forall i_k \in \mathcal{T}^k,$$

(6.3) $$j_k = new^{k,l}(i_k) \Rightarrow part^k(i_k) = part^k(j_k)$$

while synchronizing events always cause at least one partitioning function to change value:

$$\forall e_l \in \mathcal{E}^*,\; \forall i_{[1,K]} \in \mathcal{T},$$

(6.4) $$j_{[1,K]} = new(e_l, i_{[1,K]}) \Rightarrow \exists k,\; part^k(i_k) \neq part^k(j_k)$$

7. Block-partition induced on R. The partitions of the local state spaces just introduced can be used to decompose the matrices $\hat{\mathbf{R}}$ and $\mathbf{R}$ into blocks $\mathbf{B}$ corresponding to the sets of states identified by the tuples $p_{[1,K]} \in \mathcal{R}$.

7.1. Perfect partition. In this section, we assume that the user-defined partitioning functions satisfy constraints (6.3) and (6.4). In other words, we assume all and only synchronizing transitions change the load parameter of the submodels. This does not imply a loss of generality, since we can always treat a local transition as if it were a synchronizing one. Thus, the matrices $\mathbf{R}^k$ and $\mathbf{W}^{k,l}$ can be block-partitioned as

$$\begin{bmatrix} \mathbf{R}^{k:0} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{R}^{k:1} & & \mathbf{0} \\ \vdots & & \ddots & \vdots \\ \mathbf{0} & \cdots & \mathbf{0} & \mathbf{R}^{k:P_k-1} \end{bmatrix}$$

and

$$\begin{bmatrix} \mathbf{W}^{k,l:0,0} & \mathbf{W}^{k,l:0,1} & \cdots & \mathbf{W}^{k,l:0,P_k-1} \\ \mathbf{W}^{k,l:1,0} & \mathbf{W}^{k,l:1,1} & & \mathbf{W}^{k,l:1,P_k-1} \\ \vdots & & \ddots & \vdots \\ \mathbf{W}^{k,l:P_k-1,0} & \mathbf{W}^{k,l:P_k-1,1} & \cdots & \mathbf{W}^{k,l:P_k-1,P_k-1} \end{bmatrix},$$

respectively, where

- The off-diagonal blocks in $\mathbf{R}^k$ are zero due to constraint (6.3).
- $\mathbf{R}^{k:p} = \mathbf{R}^k_{\mathcal{T}^{k:p},\mathcal{T}^{k:p}}$ and
- $\mathbf{W}^{k,l:p,p'} = \mathbf{W}^{k,l}_{\mathcal{T}^{k:p},\mathcal{T}^{k:p'}}$.

Just as we defined a lexicographic order Ψ on $\mathcal{T}$, we can define one on the tuples in $\mathcal{R}$:

$$\Omega : \mathcal{R} \to \{0, \ldots, |\mathcal{R}| - 1\}.$$

Then, the transition matrix $\mathbf{R}$ can be rewritten in a block-structured fashion as

$$\mathbf{R} = \begin{bmatrix} \mathbf{B}^0 & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{B}^1 & & \mathbf{0} \\ \vdots & & \ddots & \vdots \\ \mathbf{0} & \cdots & \mathbf{0} & \mathbf{B}^{|\mathcal{R}|-1} \end{bmatrix} +$$

(7.1)
$$\begin{bmatrix} \mathbf{0} & \mathbf{B}^{0,1} & \cdots & \mathbf{B}^{0,|\mathcal{R}|-1} \\ \mathbf{B}^{1,0} & \mathbf{0} & & \mathbf{B}^{1,|\mathcal{R}|-1} \\ \vdots & & \ddots & \vdots \\ \mathbf{B}^{|\mathcal{R}|-1,0} & \mathbf{B}^{|\mathcal{R}|-1,1} & \cdots & \mathbf{0} \end{bmatrix},$$

where the blocks are defined as

- $\mathbf{B}^b = \bigoplus_{k=1}^{K} \mathbf{R}^{k:p_k} + \sum_{l=1}^{L} \lambda_l \bigotimes_{k=1}^{K} \mathbf{W}^{k,l:p_k,p_k}$,
- $\mathbf{B}^{b,b'} = \sum_{l=1}^{L} \lambda_l \bigotimes_{k=1}^{K} \mathbf{W}^{k,l:p_k,p'_k}$,

and $b = \Omega(p_{[1,K]})$ and $b' = \Omega(p'_{[1,K]})$. Note that the off-diagonal blocks of the first term in the right-hand-side of Eq. 7.1 are zero because of the structure of $\mathbf{R}^k$, while the diagonal blocks of the second term are zero because of constraint (6.4). If our assumptions were violated, we simply would have to allow for the possibility that these blocks might be nonzero instead. Note also that rectangular matrices are involved in the Kronecker product of matrices $\mathbf{W}^{k,l:p_k,p'_k}$, unlike Eq. 4.4, which uses only square matrices. However, this does not impose any restriction or complication on the solution algorithm.

Continuing with our fork-and-join example of Fig. 6.1, the blocks of the transition matrices $\mathbf{R}^k$ and $\mathbf{W}^{k,l}$ are

$$\mathbf{R}^{k:0} = [0],\ \mathbf{R}^{k:1} = \begin{bmatrix} 0 & 0 \\ \lambda_k & 0 \end{bmatrix},\ \mathbf{R}^{k:2} = \begin{bmatrix} 0 & 0 & 0 \\ \lambda_k & 0 & 0 \\ 0 & \lambda_k & 0 \end{bmatrix},$$

$$\mathbf{W}^{k,0:0,1} = \begin{bmatrix} 0 & 1 \end{bmatrix}, \quad \mathbf{W}^{k,0:1,2} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

$$\mathbf{W}^{k,3:1,0} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad \mathbf{W}^{k,3:2,1} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}.$$

Then,

$$\mathbf{R} = \begin{bmatrix} \mathbf{B}^0 & \mathbf{B}^{0,1} & \mathbf{0} \\ \mathbf{B}^{1,0} & \mathbf{B}^1 & \mathbf{B}^{1,2} \\ \mathbf{0} & \mathbf{B}^{2,1} & \mathbf{B}^2 \end{bmatrix} =$$

$$\left[\begin{array}{c|cccc|ccccccccc}
0 & 0 & 0 & 0 & \lambda_0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\hline
\lambda_3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \lambda_0 & 0 & 0 & 0 & 0 \\
0 & \lambda_2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \lambda_0 & 0 & 0 & 0 \\
0 & \lambda_1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \lambda_0 & 0 \\
0 & 0 & \lambda_1 & \lambda_2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \lambda_0 \\
\hline
0 & \lambda_3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & \lambda_3 & 0 & 0 & \lambda_2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & \lambda_2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & \lambda_3 & 0 & \lambda_1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & \lambda_3 & 0 & \lambda_1 & 0 & \lambda_2 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & \lambda_1 & 0 & \lambda_2 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \lambda_1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \lambda_1 & 0 & \lambda_2 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \lambda_1 & 0 & \lambda_2 & 0
\end{array}\right] \tag{7.2}$$

As we can easily check, all 14 reachable states of $\mathcal{T}$ are properly addressed and no unreachable state is created by the Kronecker products within the blocks of matrix $\mathbf{R}$.

We also observe that the block-tridiagonal form of $\mathbf{R}$ is due to the fact that, in our case, synchronizing events can only decrement or increment the value of the local partitioning functions by one, a fairly common occurrence that could be further exploited in implementations of our method.

7.2. Imperfect partition. In the case of imperfect partitioning, the blocks for the transition matrix $\mathbf{R}$ in Eq. (7.1) must be restricted to the sets of reachable states:

- $\left(\mathbf{B}^{b}\right)_{\mathcal{T}\cap\times_{k=1}^{K}\mathcal{T}^{k:p_k},\mathcal{T}\cap\times_{k=1}^{K}\mathcal{T}^{k:p_k}}$,
- $\left(\mathbf{B}^{b,b'}\right)_{\mathcal{T}\cap\times_{k=1}^{K}\mathcal{T}^{k:p_k},\mathcal{T}\cap\times_{k=1}^{K}\mathcal{T}^{k:p'_k}}$,

with $b = \Omega(p_{[1,K]})$ and $b' = \Omega(p'_{[1,K]})$, as before.

Given again the fork-and-join model of Fig. 6.1, assume that we partition each local state space $\mathcal{T}^k$, $k = 1, 2$ into $P_k = 2$ classes as:

$$\mathcal{T}^{k:0} = \{(002), (011), (101)\}$$
$$\mathcal{T}^{k:1} = \{(020), (110), (200)\}$$

This partition is imperfect since $\mathcal{T}^{k:0}$ includes local states with different local loads (values of $\#(a_{k3})$ for submodel m_k), while it is necessary to impose the global constraint $\#(a_{13}) = \#(a_{23})$ to achieve perfect partitioning (this is clearly not a good choice, given the existence of a perfect partition, but it is useful for illustrative purposes).

The blocks of the transition matrices $\mathbf{R}^k$ and $\mathbf{W}^{k,l}$, for $k = 1, 2$, are then:

$$\mathbf{R}^{k:0} = \begin{bmatrix} 0 & 0 & \lambda_0 \\ \lambda_3 & 0 & 0 \\ 0 & \lambda_k & 0 \end{bmatrix}, \quad \mathbf{R}^{k:1} = \begin{bmatrix} 0 & 0 & 0 \\ \lambda_k & 0 & 0 \\ 0 & \lambda_k & 0 \end{bmatrix},$$

$$\mathbf{W}^{k,0:0,0} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad \mathbf{W}^{k,3:0,0} = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix},$$

$$\mathbf{W}^{k,0:0,1} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{W}^{k,3:1,0} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}.$$

With the new choice of partition for the local state spaces, we obtain $\mathcal{R} = \{(0,0), (1,1)\}$. Then, if we restrict the blocks of matrix

$$\begin{bmatrix} \mathbf{B}^0 & \mathbf{B}^{0,1} \\ \mathbf{B}^{1,0} & \mathbf{B}^1 \end{bmatrix}$$

to the reachable states only, we obtain the same matrix $\mathbf{R}$ as in Eq. (7.2), but with coarser blocks. More precisely, the original blocks define 9×9 matrices, which are reduced to a 5×5 matrix for block $\mathbf{B}^0$, a 5×9 matrix for $\mathbf{B}^{0,1}$, and a 9×5 matrix for block $\mathbf{B}^{1,0}$, while $\mathbf{B}^1$ remains a 9×9 matrix. Even if imperfect, this partition is still more memory-efficient than working directly on the potential state space of size $|\hat{\mathcal{T}}| = 36$.

8. Equations for the solution of the CTMC. We discuss first the perfect partition case. An alternative way of writing Eq. 3.1 in a block-structured way is

$$\boldsymbol{\pi}^b \cdot (\mathbf{B}^b - diag(\mathbf{h}^b)^{-1}) + \sum_{b' \neq b} \boldsymbol{\pi}^{b'} \cdot \mathbf{B}^{b',b} = \mathbf{0}$$

$$\text{and} \qquad \sum_{b} \boldsymbol{\pi}^b \cdot \mathbf{1}_{|\mathcal{T}\cap\times_{k=1}^{K}\mathcal{T}^{k:p_k}|\times 1} = 1,$$

where $\boldsymbol{\pi} = [\boldsymbol{\pi}^0, \ldots, \boldsymbol{\pi}^{|\mathcal{R}|-1}]$ and $\mathbf{h} = [\mathbf{h}^0, \ldots, \mathbf{h}^{|\mathcal{R}|-1}]$ are the corresponding block-partition of $\boldsymbol{\pi}$ and of $\mathbf{h}$, the vector of expected holding times, and, again, $b = \Omega(p_{[1,K]})$ and $b' = \Omega(p'_{[1,K]})$. Assuming that the diagonal of $\mathbf{R}$, hence of every block $\mathbf{B}^b$, is zero, $\mathbf{h}$ satisfies

$$diag(\mathbf{h}) = rwsm(\mathbf{R})^{-1}.$$

Its blocks $\mathbf{h}^b \in I\!R^{|\mathcal{T} \cap \times_{k=1}^{K} \mathcal{T}^{k:p_k}|}$, $b = 0, \ldots, |\mathcal{R}| - 1$, can be computed as:

$$\mathbf{h}^b = \left(\mathbf{B}^b \cdot \mathbf{1} + \sum_{b' \neq b} \mathbf{B}^{b,b'} \cdot \mathbf{1} \right)^{-1}.$$

These blocks can be explicitly stored to speed up the computation, instead of recomputing them at each iteration.

Since even the blocks $\mathbf{B}^b$ tend to be very large for practical modeling problems, indirect iterative numerical methods based on this partition should be applied. In general, such block-iterative methods require more computation time per iteration, but they exhibit a good rate of convergence. The iterative method starts with an initial guess $\boldsymbol{\pi}[0]$ and computes successive vectors $\boldsymbol{\pi}[m]$, until the desired convergence is reached. The Gauss-Seidel method can then be written in a blockwise manner as:

$$\forall b = 0, \ldots, |\mathcal{R}| - 1, \quad \boldsymbol{\pi}^b[m+1] \qquad \Big(\boldsymbol{\pi}^b[m] \cdot \mathbf{B}^b +$$

$$\sum_{b'=0}^{b-1} \boldsymbol{\pi}^{b'}[m+1] \cdot \mathbf{B}^{b',b} + \sum_{b'=b+1}^{|\mathcal{R}|-1} \boldsymbol{\pi}^{b'}[m] \cdot \mathbf{B}^{b',b} \Big) \cdot \mathbf{h}^b.$$

or, for the block Gauss-Seidel method with overrelaxation (SOR) ω:

$$\boldsymbol{\pi}^b[m+1] \qquad (1-\omega)\boldsymbol{\pi}^b[m] + \omega \cdot \Big(\boldsymbol{\pi}^b[m] \cdot \mathbf{B}^b +$$

$$\sum_{b'=0}^{b-1} \boldsymbol{\pi}^{b'}[m+1] \cdot \mathbf{B}^{b',b} + \sum_{b'=b+1}^{|\mathcal{R}|-1} \boldsymbol{\pi}^{b'}[m] \cdot \mathbf{B}^{b',b} \Big) \cdot \mathbf{h}^b.$$

For the choice of the relaxation parameter ω, $0 < \omega \leq 2$, which affects the convergence rate, we refer to [20].

The analogous equations for the Jacobi method are

$$\boldsymbol{\pi}^b[m+1] \qquad \Big(\boldsymbol{\pi}^b[m] \cdot \mathbf{B}^b + \sum_{b'=0}^{|\mathcal{R}|-1} \boldsymbol{\pi}^{b'}[m] \cdot \mathbf{B}^{b',b} \Big) \cdot \mathbf{h}^b.$$

and, if relaxation is used,

$$\boldsymbol{\pi}^b[m+1] \qquad (1-\omega)\boldsymbol{\pi}^b[m] + \omega \cdot \Big(\boldsymbol{\pi}^b[m] \cdot \mathbf{B}^b +$$

$$\sum_{b'=0}^{|\mathcal{R}|-1} \boldsymbol{\pi}^{b'}[m] \cdot \mathbf{B}^{b',b} \Big) \cdot \mathbf{h}^b.$$

For an imperfect partition, elements of the partial probability vectors $\boldsymbol{\pi}^b$, $\quad b = 0, \ldots, |\mathcal{R}| - 1$, have to be additionally addressed by the corresponding index functions

$$\Psi^b : \mathcal{T} \cap \times_{k=1}^{K} \mathcal{T}^{k:p_k} \rightarrow \{0, \ldots, |\mathcal{T} \cap \times_{k=1}^{K} \mathcal{T}^{k:p_k}| - 1\}.$$

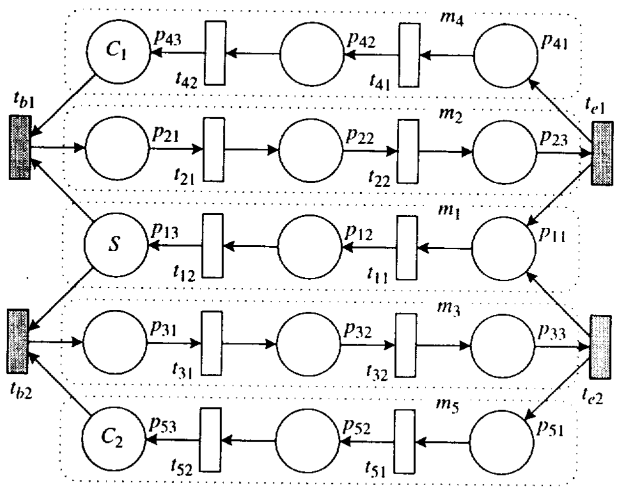

FIG. 9.1. *A model of a tasking system.*

The index functions Ψ^b impose a lexicographic order.

In other words, for an imperfect partition, we must store the level-2 boolean vectors and use them to test whether certain states are reachable. However, if we accept an iteration vector $\tilde{\pi}$ larger than π (but smaller than the $\hat{\pi}$ for the entire potential state space), we can simply consider all the states described by the level-1 boolean vector; some of these are reachable and some are not, since we ignore the level-2 information (indeed, we do not even have to store the level-2 vectors). The approach is similar to the original one, suggested by Plateau-Stewart [20], but uses only a subset $\tilde{\mathcal{T}}$ of the potential state space: $\mathcal{T} \subseteq \tilde{\mathcal{T}} \subseteq \hat{\mathcal{T}}$.

We can then use the formulas for the perfect partitioning and iterate disregarding whether the states corresponding to a $p_{[1,K]} \in \mathcal{R}$ are reachable or not. If, as in Plateau-Stewart, we ensure that any entry corresponding to an unreachable state is set to zero in the initial probability vector $\pi^b[0]$, the nonzero entries of the final iteration vector $\pi^b[m]$ after convergence will be the probabilities of the reachable states. For our fork-and-join model of Fig. 6.1, this results in a CTMC with 18 states, more than the required 14, but still much less than the potential 36 states.

9. Example and timing results. For the numerical computation we consider the distributed tasking system described by the GSPN shown in Fig. 9.1.

It consists of S server tasks and two classes of customer tasks, C_1 of class 1 and C_2 of class 2. Five submodels m_k, $k = 1, \ldots, 5$, indicated by dashed boxes, interact through the four synchronizing transitions shown in grey, representing the beginning (t_{b1} and t_{b2}) and the end (t_{e1} and t_{e2}) of a rendezvous between a server task and a task of class 1 or 2, respectively. Tasks of each class run their local computation independently, until they need to synchronize with a server task. If no server task is ready to synchronize with them, they simply wait until one becomes ready.

The parameters specifying the stochastic behavior of this software system are the rates of each transition in the GSPN. We assume that the weight of every transition is constant, this implies that each transition represents an independent hardware resource with single-server semantics, that is, without internal parallelism (e.g., a single processing unit). Hence, we only need to specify λ_x for each transition t_x. More complex dependencies can be accommodated by our model:

- If, for each submodel m_k, t_{k1} and t_{k2} share the same hardware resource, this could be easily accommodated, since both transitions are local to the same model m_k. Only the entries of $\mathbf{R}^k$ would be affected.

- If a transition in submodel m_k represents a parallel hardware resource, this can also be reflected in matrix $\mathbf{R}^k$, by appropriately scaling the rates according to the load on that transition in each local state.

However, the state-dependent behavior across submodels is more limited:

- The rate of each synchronizing transition can only depend in a "product-form" fashion on the local states of the involved submodels. For example, the rate of t_{b1} can be of the form $\lambda_{b1} \cdot f(\#(p_{13})) \cdot g(\#(p_{43}))$. This cannot describe the case where the work performed by t_{b1} to initiate all the waiting rendezvous (there are $\min\{\#(p_{13}), \#(p_{43})\}$ of them) can be performed in parallel on multiple processors.
- Analogously, if the servers run on a single processor and the rendezvous actions are performed on them, submodels m_1, m_2, and m_3 must be merged into a single submodel m_{123}. Only then we can correctly represent the processor-sharing interactions between all the transitions using the processor, in the local matrix for this larger submodel, $\mathbf{R}^{123}$.
- It should be noted that, in either case, the shared use of resource needed by the synchronizing transitions is still not represented correctly, since, again, we cannot express it in product-form fashion. For example, the model cannot represent the fact that, when $\#(p_{13}) = s$, $\#(p_{43}) = c_1$, and $\#(p_{53}) = c_2$, there are $\min\{s, c_1 + c_2\}$ rendezvous initiations sharing the same processor. The net effect is that the total rate of t_{b1} and t_{b_2} will be twice what it should be in any global state where they are both enabled. However, the timing of these synchronizing transitions should be much faster than that of the local transitions in practical models. Indeed, it might be appropriate to use immediate transitions to model the synchronizations in our model. We have shown how the Kronecker-based approach can accommodate such immediate synchronizations in [9].

The numerical values of the rates of the synchronizing transitions are $\lambda_{b1} = \lambda_{b2} = \lambda_{e1} = \lambda_{e2} = 10.0$, while the rates of the local transitions are $\lambda_{k1} = \lambda_{k2} = k$ for $k = 1, 2, 3, 4, 5$. We assume that there are N tasks of each type, $N = C_1 = C_2 = S$.

In Table 9.1, we give the expected number E_k of tasks in each submodel as a function of the initial number of tokens N. In particular, tokens in submodels m_2 and m_3 represent tasks of class 1 and 2 in rendezvous with a server, respectively. We also compute the throughput τ of the system, defined as the expected firing rate of any local transition in m_1, that is, the rate of completion of server tasks.

We now comment on the memory and execution complexity of our approach. We consider two different structured configuration of the tasking system.

- MOD$_5$, a model consisting of five submodels m_1, m_2, m_3, m_4, and m_5.
- MOD$_3$, a model consisting of three submodels: a larger submodel m_{123} obtained by merging submodels m_1, m_2, and m_3, plus the submodels m_4 and m_5.

We compute the steady-state solution of MOD$_5$ using the conventional structured approach based on the "actual" state space $\mathcal{T}$ (CS$_5^{act}$), the "potential" state space $\hat{\mathcal{T}}$ (CS$_5^{pot}$), or our approach based on a perfect partition (PP$_5$). For MOD$_3$, we use either the conventional structured approach, again based on the actual or potential state space $\mathcal{T}$ (CS$_3^{act}$ and CS$_3^{pot}$), or our imperfect partition approach (IP$_3$) based on $\hat{\mathcal{T}}$. For PP$_5$, we partition the local state space of submodels $m_k, k = 1, \ldots, 5$ according to their population, through the function $part^k = \#(p_{k1}) + \#(p_{k2}) + \#(p_{k3})$. For IP$_3$, we partition the local state space of submodels m_4 and m_5 as for PP$_5$, while that of submodel m_{123} is partitioned according to $part^{123} = \#(p_{11}) + \#(p_{12}) + \#(p_{13})$.

N	E_1	E_2	E_3	E_4	E_5	τ
1	0.688	0.182	0.130	0.818	0.870	0.335
2	1.466	0.316	0.218	1.684	1.782	0.539
3	2.312	0.412	0.276	2.588	2.724	0.663
4	3.207	0.478	0.315	3.522	3.685	0.741
5	4.134	0.524	0.342	4.476	4.658	0.792
6	5.083	0.557	0.360	5.443	5.640	0.827
7	6.046	0.580	0.374	6.420	6.626	0.853
8	7.019	0.598	0.384	7.402	7.616	0.872
9	8.001	0.613	0.390	8.387	8.610	0.886

TABLE 9.1
Expected population in each submodel and throughput.

N	$\lvert\mathcal{T}^k\rvert$	$\lvert\mathcal{T}\rvert$	$\lvert\hat{\mathcal{T}}\rvert$
1	4	45	1,024
2	10	693	100,000
3	20	6,060	3,200,000
4	35	36,981	52,521,875
5	56	175,383	550,731,776
6	84	689,325	$4.182 \cdot 10^9$
7	120	2,341,404	$2.488 \cdot 10^{10}$
8	165	7,074,990	$1.222 \cdot 10^{11}$
9	220	19,421,038	$5.154 \cdot 10^{11}$

TABLE 9.2
Size of the state space for MOD$_5$ ($k = 1, \ldots, 5$).

The global p-invariants of the tasking system are (no local p-invariants exist):

$$\sum_{k=1}^{3} \#(p_{k1}) + \#(p_{k2}) + \#(p_{k3}) = S$$

$$\sum_{k=2,4} \#(p_{k1}) + \#(p_{k2}) + \#(p_{k3}) = C_1$$

$$\sum_{k=3,5} \#(p_{k1}) + \#(p_{k2}) + \#(p_{k3}) = C_2.$$

The perfect partition of model PP$_5$ distinguishes among the local loads $\#(p_{k1}) + \#(p_{k2}) + \#(p_{k3})$ for each submodel m_k; this naturally correspond to the enforcement of the global p-invariants, whereas partition IP$_3$ fails to do so, because the loads $\#(p_{21}) + \#(p_{22}) + \#(p_{23})$ and $\#(p_{31}) + \#(p_{32}) + \#(p_{33})$ cannot be uniquely determined given a class in the partition of submodel m_{123}.

In the following, we compare the complexity of the six solutions CS$_5^{act}$, CS$_5^{pot}$, PP$_5$, CS$_3^{act}$, CS$_3^{pot}$, and IP$_3$. The size of the local state spaces $\mathcal{T}^k$ and of the global state spaces $\mathcal{T}$ and $\hat{\mathcal{T}}$ are given for configuration MOD$_5$ in Table 9.2. In the tasking model, the two rendezvous create unreachable states (any marking where the sum of tokens in in m_1, m_2, and m_3 is not equal to S), so that $|\mathcal{T}| < |\hat{\mathcal{T}}|$. In addition, the total population of m_2 and m_4 must equal C_1, and the total population of m_3 and m_5 must equal C_2. Hence, any

N	$\lvert\mathcal{T}^{123}\rvert$	$\lvert\mathcal{T}\rvert$	$\lvert\tilde{\mathcal{T}}\rvert$	$\lvert\hat{\mathcal{T}}\rvert$
1	9	45	63	144
2	45	693	1,305	4,500
3	165	6,060	14,504	66,000
4	495	36,981	107,811	606,375
5	1,287	175,383	603,855	4,036,032
6	3,003	689,325	2,739,443	21,189,168
7	6,435	2,341,404	10,552,950	92,664,000
8	12,870	7,074,990	42,742,692	$3.5038 \cdot 10^8$
9	24,310	19,421,038	108,301,458	$1.1766 \cdot 10^9$

TABLE 9.3
Size of the state space for MOD$_3$ ($k = 4, 5$).

N	nz$_3$	$\lvert\mathcal{R}\rvert_3$	nz$_5$	$\lvert\mathcal{R}\rvert_5$	$\eta(\mathbf{R})$
1	18	3	32	3	90
2	122	6	88	6	2,376
3	530	10	220	10	27,432
4	1,810	15	440	15	199,932
5	5,230	21	770	21	1,075,158
6	13,150	28	1,232	28	4,644,900
7	30,702	36	1,848	36	16,991,520
8	65,310	45	2,640	45	54,513,576
9	129,360	55	3,630	55	157,241,916

TABLE 9.4
Memory requirements.

population vector not fulfilling these global p-invariants is unreachable as well, another reason for having $|\mathcal{T}| < |\hat{\mathcal{T}}|$. Analogously, the size of the local state spaces $\mathcal{T}^{123}$ and of the global state spaces $\mathcal{T}$, $\tilde{\mathcal{T}}$, and $\hat{\mathcal{T}}$, are given for configuration MOD$_3$ in Table 9.3 ($\mathcal{T}^4$ and $\mathcal{T}^5$ are as in Table 9.2).

The overall number of nonzero elements for the sparse storage of the matrices $\mathbf{R}^k$ and $\mathbf{W}^{k,j}$ are given in Table 9.4 for configurations MOD$_3$ and MOD$_5$, under the column headings nz$_3$ and nz$_5$, respectively, together with the size of $\mathcal{R}$ in the two cases. For comparison, we also list the number of nonzero elements in $\mathbf{R}$, which would have to be explicitly generated and stored for conventional unstructured solution methods. One can see that a conventional solution not based on structured methods would fail for $N \geq 8$ because of the memory required to store $\mathbf{R}$ explicitly.

Instead, the memory requirements for the local matrices are negligible with respect to the storage of the iteration vectors. The main practical limitation of the structured approach is then memory for these vectors[1] and for the state space (needed only by CS$_5^{act}$ and CS$_3^{act}$). The iteration vectors are of size $|\mathcal{T}|$ for CS$_5^{act}$, CS$_3^{act}$, and PP$_5$; of size $|\hat{\mathcal{T}}|$ for CS$_5^{pot}$ and CS$_3^{pot}$ (but MOD$_5$ and MOD$_3$ have different potential state spaces); and of size $|\tilde{\mathcal{T}}|$ for IP$_3$. Hence, the memory requirements for these vectors exceed the available

[1]Two iteration vectors, $\boldsymbol{\pi}[m]$ and $\boldsymbol{\pi}[m+1]$, are needed for the Jacobi method or for the Gauss-Seidel method when the stopping criterion is based on a relative or absolute comparison of two successive iteration vectors. Only one vector is instead needed with Gauss-Seidel if we use the norm of the residual $\boldsymbol{\pi}[m] \cdot (\mathbf{R} - rwsm(\mathbf{R}))$ as a criterion.

]

N	CS_5^{act}	CS_5^{pot}	PP_5	CS_3^{act}	CS_3^{pot}	IP_3
1	0.117 (131)	0.107 (31)	0.050 (117)	0.083 (131)	0.063 (131)	0.050 (117)
2	0.800 (57)	0.700 (57)	0.633 (62)	0.567 (57)	0.503 (57)	0.450 (62)
3	11.917 (86)	10.533 (86)	7.917 (77)	8.167 (86)	7.150 (86)	6.267 (77)
4	114.967 (126)	— (—)	77.867 (114)	82.450 (126)	72.833 (126)	55.400 (114)
5	821.517 (171)	— (—)	551.733 (155)	620.717 (171)	525.900 (171)	408.550 (155)
6	4,282.368 (219)	— (—)	3,097.833 (199)	3,345.683 (219)	— (—)	2,241.867 (199)
7	— (—)	— (—)	13,153.000 (245)	14,721.867 (270)	— (—)	10,121.867 (245)
8	— (—)	— (—)	47,021.100 (294)	52,324.993 (325)	— (—)	— (—)
9	— (—)	— (—)	157,960.610 (346)	— (—)	— (—)	— (—)

TABLE 9.5
Time (iterations) for the numerical solution as a function of N.

memory and cause the solution to fail first for CS_5^{pot} (when $N \geq 4$), then for CS_3^{pot} (when $N \geq 6$), and finally for IP_3 (when $N \geq 8$), while CS_3^{act} and PP_5 can be solved up to $N = 9$. However, CS_3^{act} experiences excessive paging due to the additional storage for the actual state space $\mathcal{T}$, so we provide data only for PP_5. In principle, CS_5^{act} could be solved also for $N = 9$, but our prototype implementation uses four-bytes (unsigned) integers to store a potential state to be searched, so it fails when $|\hat{\mathcal{T}}| \geq 2^{32} = 4.494 \cdot 10^9$, which happens for $N \geq 7$, in Table 9.2.

The overall solution time has two components: the time to explore the reachability set (only CS_5^{act} and CS_3^{act} require this step) and the time for the numerical solution. Since the former is negligible in comparison to the latter, Table 9.5 reports only the numerical solution time, in seconds, for the six approaches. We use the Jacobi method with a relaxation parameter 0.9 for the conventional structured methods, or a block-Jacobi method for PP_5 and IP_3, with the same relaxation parameter. Table 9.5 also lists the number of iterations required for the solution. In the conventional structured case, these number are the same across the four models, since they differ only in the algorithmic implementation, but they otherwise perform exactly the same floating point operations in the same order on the relevant entries of the iteration vectors. Using a block Jacobi method for PP_5 and IP_3 models, both the number of outer iterations and of inner iterations would be relevant. However, for a fair comparison with the conventional structured solution, we limit the number of inner iterations to one. Increasing this number would reduce the number of outer iterations and result, in most cases, in a substantial speedup. The optimization of the block Jacobi method by varying the number of inner iterations, the relaxation, the order in which blocks are considered, or using modified adaptive methods are beyond the scope of this paper.

The uniform distribution is the initial guess for $\pi[0]$, and the relative convergence criterion

$$\max_{i \in \mathcal{T}} \left\{ \frac{|\pi[m]_i - \pi[m+1]_i|}{\pi[m+1]_i} \right\} < 10^{-3}$$

is used to stop the iterations. The program is run on a workstation with a 400 Mhz DEC Alpha 21164 processor and 256 MByte of main memory. In all cases, the vector $\mathbf{h}$ is stored explicitly and all vectors are stored in double precision. The memory requirements would be approximately halved had we used single precision instead.

As we observed before, IP_3 cannot be solved for $N \geq 8$ because of the size of the state space $|\hat{\mathcal{T}}|$, given in Table 9.3. However, when it can be run, IP_3 is the fastest method (Table 9.5). It is faster than PP_5, since

Kronecker products of only three matrices, instead of five, are required, and it is faster than CS_3, because no logarithmic search to compute the index of reachable states is necessary. For $N \geq 8$, PP_5 is the fastest method, and for $N \geq 9$ it is the only feasible method. However, using the multilevel data structure of [8] for the storage of the actual state space $\mathcal{T}$ instead of the unsigned integer vector we used in our prototype implementation, CS_3^{act} and CS_5^{act} would also be feasible.

For the perfect partition PP_5 and for the imperfect partition IP_3 using iteration vectors of size $|\tilde{\mathcal{T}}|$, or for the conventional structured methods based on the potential states space, CS_5^{pot} and CS_3^{pot}, it is not necessary to explore and store the global state space. Instead, the local state spaces $\mathcal{T}^k$ are explored and partitioned subject to the partitioning function, and the relation $\mathcal{R}$ is determined exploring the global state space of an aggregated stochastic automatic network that fulfills all global p-invariants. In other special cases (e.g., the SGSPNs [10, 11]), the submodels are restricted in such a way that the local state spaces $\hat{\mathcal{T}}^k$ can be obtained in isolation, thus avoiding the reachability set exploration altogether, but the resulting $\hat{\mathcal{T}}^k$ might then be a strict superset of $\mathcal{T}^k$. In full generality, though, none of these approaches might be possible, and the local state spaces $\mathcal{T}^k$ must be obtained by projection of the global state space on the k-th component. In this case, we can delete the data structure used to store $\mathcal{T}$ as soon as the sets $\mathcal{T}^k$ have been computed. For CS_5^{act} and CS_3^{act}, we must instead keep $\mathcal{T}$ throughout the numerical solution, to compute the indexing function Ψ.

We stress that, to achieve a fair comparison of the various approaches, we used the Jacobi method in all cases, since it effectively allows us to ignore the rows and columns of $\hat{\mathbf{R}}$ corresponding to the unreachable states. This is essential for CS_5^{pot}, CS_3^{pot}, and IP_3 (if based on $\tilde{\mathcal{T}}$). However, any approach based on the actual state space can also use a SOR-type iteration, which has faster convergence, although the cost per iteration can be somewhat larger in this case, if spurious entries in $\hat{\mathbf{R}}_{\hat{\mathcal{T}}\setminus\mathcal{T},\mathcal{T}}$ need to be recognized and ignored explicitly. Numerical experiments show that the overall solution time is then generally lower for CS_5^{act}, CS_3^{act}, and IP_3 (if based instead on $\mathcal{T}$), but certainly lower for a (block) SOR applied to PP_5, which does not suffer from the problem of unreachable states.

10. Conclusion. We presented a new approach for the Kronecker-based analysis of structured continuous-time Markov models. By partitioning the "local state spaces" of each submodel, we are able to restrict the description of the "potential" transition rate matrix $\hat{\mathbf{R}}$ to the "actual" transition rate matrix $\mathbf{R}$ corresponding to the reachable states only, without having to incur a logarithmic overhead. Indeed, the partition also allows us to avoid storing the state space altogether, thus reducing the peak memory requirements.

In addition to the more straightforward "perfect" partition, we also introduced the concept of an "imperfect" partition, where the resulting state space is larger than the actual state space, but still, hopefully, much smaller than the potential one. Surprisingly, this might result in the shortest solution time, as long as the computation fits in memory.

One substantial advantage of our approach is that it can naturally employ a block SOR method for the numerical solution, which has then a faster convergence rate than the Jacobi method traditionally used in previous Kronecker-based approaches.

Our algorithms are implemented in the tool SNS [23]. For a copy of the program please contact the second author.

REFERENCES

[1] P. Buchholz, *Numerical solution methods based on structured descriptions of Markovian models*, in Computer performance evaluation, G. Balbo and G. Serazzi, eds., Elsevier Science Publishers B.V.

(North-Holland), 1991, pp. 251–267.

[2] ——, *A class of hierarchical queueing networks and their analysis*, Queueing Systems., 15 (1994), pp. 59–80.

[3] P. Buchholz, G. Ciardo, S. Donatelli, and P. Kemper, *Complexity of Kronecker operations on sparse matrices with applications to the solution of Markov models*, ICASE Report, Institute for Computer Applications in Science and Engineering, Hampton, VA. In review.

[4] P. Buchholz and P. Kemper, *Numerical analysis of stochastic marked graphs*, in Proc. 6th Int. Workshop on Petri Nets and Performance Models (PNPM'95), Durham, NC, Oct. 1995, IEEE Comp. Soc. Press, pp. 32–41.

[5] J. Campos, M. Silva, and S. Donatelli, *Structured solution of stochastic DSSP systems*, in Proc. 7th Int. Workshop on Petri Nets and Performance Models (PNPM'97), Saint Malo, France, June 1997, IEEE Comp. Soc. Press, pp. 91–100.

[6] G. Ciardo, A. Blakemore, P. F. J. Chimento, J. K. Muppala, and K. S. Trivedi, *Automated generation and analysis of Markov reward models using Stochastic Reward Nets*, in Linear Algebra, Markov Chains, and Queueing Models, C. Meyer and R. J. Plemmons, eds., vol. 48 of IMA Volumes in Mathematics and its Applications, Springer-Verlag, 1993, pp. 145–191.

[7] G. Ciardo, J. Gluckman, and D. Nicol, *Distributed state-space generation of discrete-state stochastic models*, INFORMS J. Comp. To appear.

[8] G. Ciardo and A. S. Miner, *Storage alternatives for large structured state spaces*, in Proc. 9th Int. Conf. on Modelling Techniques and Tools for Computer Performance Evaluation, R. Marie, B. Plateau, M. Calzarossa, and G. Rubino, eds., LNCS 1245, Saint Malo, France, June 1997, Springer-Verlag, pp. 44–57.

[9] G. Ciardo and M. Tilgner, *On the use of Kronecker operators for the solution of generalized stochastic Petri nets*, ICASE Report 96-35, Institute for Computer Applications in Science and Engineering, Hampton, VA, May 1996.

[10] S. Donatelli, *Superposed Stochastic Automata: a class of stochastic Petri nets amenable to parallel solution*, in Proc. 4th Int. Workshop on Petri Nets and Performance Models (PNPM'91), Melbourne, Australia, Dec. 1991, IEEE Comp. Soc. Press, pp. 54–63.

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

[12] P. Kemper, *Numerical analysis of superposed GSPNs*, in Proc. 6th Int. Workshop on Petri Nets and Performance Models (PNPM'95), Durham, NC, Oct. 1995, IEEE Comp. Soc. Press, pp. 52–61.

[13] T. Murata, *Petri Nets: properties, analysis and applications*, Proc. of the IEEE, 77 (1989), pp. 541–579.

[14] S. Pissanetzky, *Sparse Matrix Technology*, Academic Press, 1984.

[15] B. Plateau, *On the stochastic structure of parallelism and synchronisation models for distributed algorithms*, in Proc. 1985 ACM SIGMETRICS Conf. on Measurement and Modeling of Computer Systems, Austin, TX, USA, May 1985, pp. 147–153.

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

[17] B. Plateau and J.-M. Fourneau, *A methodology for solving Markov models of parallel systems*, J.

Par. and Distr. Comp., 12 (1991), pp. 370–387.

[18] B. PLATEAU, J.-M. FOURNEAU, AND K. H. LEE, *PEPS: a package for solving complex Markov models of parallel systems*, in Proc. 4th Int. Conf. Modelling Techniques and Tools, R. Puigjaner, ed., 1988, pp. 341–360.

[19] B. PLATEAU AND W. J. STEWART, *Stochastic automata networks: product forms and iterative solutions*, Rapports de Recherche 2939, INRIA, July 1996.

[20] W. J. STEWART, *Introduction to the Numerical Solution of Markov Chains*, Princeton University Press, 1994.

[21] W. J. STEWART, K. ATIF, AND B. PLATEAU, *The numerical solution of stochastic automata networks*, Europ. J. of Oper. Res., 86 (1995), pp. 503–525.

[22] Y. TAKAHASHI, *Aggregate approximation for acyclic queuing networks with communication blocking*, in Queueing Networks with Blocking, H. G. Perros and T. Altiok, eds., Elsevier Science Publishers B.V., 1989, pp. 33–47.

[23] M. TILGNER, *SNS1.0: Synchronized Network Solver — User's Manual Solver*, Technical Report Series B: Operations Research. B-316, Research Reports on Mathematical and Computing Sciences, Tokyo Institute of Technology, 1996.

[24] M. TILGNER, Y. TAKAHASHI, AND G. CIARDO, *SNS 1.0: Synchronized Network Solver*, in 1st International Workshop on Manufacturing and Petri Nets, Osaka, Japan, June 1996, pp. 215–234.