View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Bilkent University Institutional Repository

Available online at www.sciencedirect.com

SCIENCE DIRECT®

ELSEVIER

Linear Algebra and its Applications 386 (2004) 83–109

LINEAR ALGEBRA AND ITS APPLICATIONS

www.elsevier.com/locate/laa

Block SOR for Kronecker structured representations

Peter Buchholz [a,1], Tuğrul Dayar [b,*]

[a] *Department of Computer Science, Dresden University of Technology, D-01062 Dresden, Germany*
[b] *Department of Computer Engineering, Bilkent University, TR-06800 Bilkent, Ankara, Turkey*

Received 12 August 2003; accepted 3 December 2003

Submitted by D. Szyld

Abstract

The Kronecker structure of a hierarchical Markovian model (HMM) induces nested block partitionings in the transition matrix of its underlying Markov chain. This paper shows how sparse real Schur factors of certain diagonal blocks of a given partitioning induced by the Kronecker structure can be constructed from smaller component matrices and their real Schur factors. Furthermore, it shows how the column approximate minimum degree (COLAMD) ordering algorithm can be used to reduce fill-in of the remaining diagonal blocks that are sparse LU factorized. Combining these ideas, the paper proposes three-level block successive over-relaxation (BSOR) as a competitive steady state solver for HMMs. Finally, on a set of numerical experiments it demonstrates how these ideas reduce storage required by the factors of the diagonal blocks and improve solution time compared to an all LU factorization implementation of the BSOR solver.
© 2004 Elsevier Inc. All rights reserved.

Keywords: Markov chains; Kronecker based numerical techniques; Block SOR; Real Schur factorization; COLAMD ordering

1. Introduction

The attraction for Markov chains (MCs) lies in that they provide exact results (up to computer precision) for performance or reliability measures of interest through numerical analysis. Unfortunately, Markovian modeling and analysis is liable to the

* Corresponding author. Tel.: +90-312-290-1981; fax: +90-312-266-4047.
E-mail addresses: p.buchholz@inf.tu-dresden.de (P. Buchholz), tugrul@cs.bilkent.edu.tr (T. Dayar).
[1] Tel.: +49-351-463-38360; fax: +49-351-463-38460.

0024-3795/$ - see front matter © 2004 Elsevier Inc. All rights reserved.
doi:10.1016/j.laa.2003.12.017

problem of state space explosion since it is not uncommon to encounter systems requiring millions of states in most realistic models today. Therefore, structured representations amenable to Kronecker based numerical techniques are gaining popularity. The essence of the Kronecker based approach is to model the system at hand in the form of interacting components so that its (larger) underlying MC is not generated but represented as a sum of Kronecker products of (smaller) component matrices, and its state space is given by the cross product of the state spaces of the components.

The concept of using Kronecker (or tensor) operations to define large MCs underlying structured representations appears in hierarchical Markovian models (HMMs) [6,10,13], in compositional Markovian models such as stochastic automata networks (SANs) [30–32,34] and different classes of superposed Stochastic Petri Nets (SPNs) [20,27], or in stochastic process algebras like PEPA [26]. HMMs provide one possible solution to compactly represent the transition matrices of large MCs without introducing unreachable states. An alternative to HMMs which represent state spaces compositionally without unreachable states are matrix diagrams [14] or representations for specific models as presented in [25]. In order to analyze structured Markovian models efficiently, various algorithms for vector-Kronecker product multiplication are devised [11,21,22,30] and used as kernels in iterative solution techniques proposed for HMMs [6,8], SANs [8,9,11,30,34,35] and superposed Generalized SPNs [27]. A distributed solution technique for HMMs is investigated in [12]. Although various iterative Kronecker based techniques have been formulated in this context, there is still room for improvement.

Results in [16] on the computation of the stationary vector of MCs show that block successive over-relaxation (BSOR) with judiciously chosen partitionings is a very competitive solver when compared with iterative aggregation-disaggregation (IAD) and incomplete LU (ILU) preconditioned projection methods. Iterative methods based on splittings (Jacobi, Gauss–Seidel, SOR) and their block versions are developed for SANs in [35]. Therein it is argued that the Kronecker based nature of the underlying continuous-time MC (CTMC) induces nested block partitionings, and the problem becomes that of solving multiple nonsingular linear systems whose coefficient matrices are the diagonal blocks of a particular partitioning. Results are reported with block Gauss–Seidel (BGS) and BSOR with diagonal blocks at the lowest level of the nested partitioning. Results with BGS using larger diagonal blocks at intermediate levels of the nested partitioning and utilizing BGS to solve diagonal blocks when they become too large to be LU factorized have appeared more recently in [23,24]. We name the latter kind of BGS solvers as being three-level as opposed to the usual two-level solvers [28], since in addition to the outer BSOR iteration at the first level there exists an intermediate BGS iteration at the second level which solves the larger diagonal blocks of the BSOR partitioning using smaller nested diagonal blocks.

In this paper, we present a three-level BSOR solver for HMMs, which are composed of multiple low level models (LLMs) and a high level model (HLM) that

defines the interaction among LLMs. As in SANs, the Kronecker structure of an HMM induces nested block partitionings in its underlying CTMC. Assuming that the HLM has multiple states, it suggests the partitioning at level 0; the LLMs define the nested partitionings at higher level numbers. LLM 1 defines the partitioning at level 1, LLM 2 defines the partitioning at level 2, and so on. In HMMs, diagonal blocks at a particular level of the nested partitioning are all square, but can have different orders in different HLM states. Consequently, off-diagonal blocks in the off-diagonal part of the HLM matrix need not be square. This is different than SANs in which all (diagonal and off-diagonal) blocks at each level of nested partitioning associated with the Kronecker structure are square and have the same order. Although there already exists a BSOR solver for SANs [35, pp. 175–176], the BSOR solver in this paper is the first one proposed for HMMs.

While developing an effective solver of the BSOR type, we show that in each HLM state there may be diagonal blocks with identical off-diagonal parts and diagonals differing from each other by a multiple of the identity matrix. We name such diagonal blocks as *candidate blocks*, explain how they can be detected and how they can mutu-ally benefit from a single real Schur factorization [19,33]. We give sufficient conditions for the existence of diagonal blocks with real eigenvalues and show how to check these conditions using component matrices. This result is important because in practice the real Schur factors of diagonal blocks satisfying these conditions are sparse; and, as we describe, these factors can be constructed from component matrices and their real Schur factors. We also show how fill-in of LU factorized diagonal blocks often can be reduced by using the column approximate minimum degree (COLAMD) ordering algorithm [17,18]. The particular BSOR implementation solves the diagonal blocks at the first level using block Gauss–Seidel (BGS) at the second and the methods of real Schur and LU factorizations at the third level. On a set of numerical experiments we demonstrate how these ideas can be used to reduce the storage required by the factors of the diagonal blocks at the third level and to improve the solution time compared to an all LU factorization implementation of the BSOR solver.

Section 2 recalls structured description of CTMCs using HMMs on an example. Section 3 expands on the concept of candidate blocks and shows how we take advantage of COLAMD. Section 4 considers implementation details and Section 5 discusses results of numerical experiments. We conclude in Section 6.

2. Structured description of CTMCs using HMMs

Formal treatment of HMMs can be found, for instance, in [8, pp. 387–390]. Since our aim is to analyze HMMs numerically, we introduce HMMs on a running example so as to center the discussion around nested block partitionings induced by their Kronecker structure. Hereafter, we refer to the CTMC underlying an HMM as the matrix Q. This matrix has nonnegative off-diagonal elements and diagonal elements that are negated row sums of its off-diagonal elements.

Example. We consider a typical benchmark from the literature which is introduced in [38]. The model associated with the Courier protocol in [38] is used in [7,27] and will also serve as one of our test cases in this paper. More information about it may be obtained from [2]. We name the example discussed here as *courier_small*. The HLM (see [7]), which has 7 states, describes the interaction among four LLMs. LLM 1 has 15 states, LLM 2 has 69 states, LLM 3 has 27 states, and LLM 4 has 30 states. We number all states starting from 0. We name the states of the HLM as *macrostates* and those of Q as *microstates*. The mapping between LLM states and HLM states is given in Table 1. Note that macrostates in an HLM may have different numbers of microstates when LLMs have partitioned state spaces as in this example. The microstates corresponding to each macrostate result from the cross-product of the state space partitions of LLMs that are mapped to that particular macrostate without unreachable states. Hence, we have the number of microstates in the last column of Table 1 as the product of the cardinalities of the corresponding LLM partitions.

In the particular system under consideration, six transitions denoted by t_0 through t_5 take place in the HLM and affect the LLMs. These transitions are captured by the following (7×7) HLM matrix:

$$\begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{ccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{pmatrix} t_1 & & & & t_2 & t_3 & t_2 \\ & t_1, t_5 & & t_0 & & t_4 & t_4 \\ & & & t_0 & & & \\ & & & t_5 & & t_4 & t_4 \\ t_4 & t_3 & & & t_1, t_5 & & t_0 \\ & t_2 & t_3 & t_2 & & t_1 & t_0 \\ t_4 & t_2 & & t_2, t_3 & & & t_1, t_5 \end{pmatrix} \end{array}. \tag{1}$$

To each transition t_1 through t_5 corresponds a Kronecker product of four (i.e., number of LLMs) LLM matrices. The matrices associated with those LLMs that do not participate in a transition are all identity. LLM 1 participates in t_1 with the matrix $Q_{t_1}^{(1)}$; LLM 2 participates in t_1 through t_4 with the matrices $Q_{t_1}^{(2)}$ through $Q_{t_4}^{(2)}$; LLM 3 participates in t_2 through t_5 with the matrices $Q_{t_2}^{(3)}$ through $Q_{t_5}^{(3)}$; and LLM 4 par-

Table 1
Mapping between LLM states and HLM states in *courier_small*

HLM	LLM 1	LLM 2	LLM 3	LLM 4	# of microstates								
0	0:14	19:68	0	0:29	15	.	50	.	1	.	30	=	22,500
1	0:14	0:1	7:26	0:29	15	.	2	.	20	.	30	=	18,000
2	0:14	2	6	0:29	15	.	1	.	1	.	30	=	450
3	0:14	2	7:26	0:29	15	.	1	.	20	.	30	=	9000
4	0:14	3:5	2:5	0:29	15	.	3	.	4	.	30	=	5400
5	0:14	6:18	1	0:29	15	.	13	.	1	.	30	=	5850
6	0:14	6:18	2:5	0:29	15	.	13	.	4	.	30	=	23,400

ticipates in t_5 with the matrix $Q_{t_5}^{(4)}$. In general, these matrices are very sparse and therefore held in row sparse format [34]. In this example, each of the transitions t_1 through t_5 affects exactly two LLMs. For instance, the Kronecker product associated with t_3 in element (0,5) of the HLM matrix in Eq. (1) is

$$I_{15} \otimes Q_{t_3}^{(2)}(19:68, 6:18) \otimes Q_{t_3}^{(3)}(0, 1) \otimes I_{30},$$

where I_m denotes the identity matrix of order m, $Q_{t_3}^{(2)}(19:68, 6:18)$ denotes the submatrix of $Q_{t_3}^{(2)}$ that lies between states 19 through 68 rowwise and states 6 through 18 columnwise, and $\otimes$ is the Kronecker product operator [37]. See Table 1 for the mapping of LLM states to HLM states. In this example, the rates associated with the 29 transitions in (1) are all 1.0, hence they have not been shown; however, they could very well be other real numbers. The transition rates are scalars that multiply the corresponding Kronecker products. As we explain in the next paragraph, although each LLM has a matrix associated with t_0, it is a special transition, called local, for which all but one of the LLM matrices in the corresponding Kronecker product are identity.

Other than Kronecker products due to the depicted transitions in (1), there is a Kronecker sum implicitly associated with each diagonal element of the HLM matrix. Each Kronecker sum is formed of four LLM matrices corresponding to local transition t_0. For instance, the Kronecker sum associated with element (3,3) of the HLM matrix is

$$Q_{t_0}^{(1)} \oplus Q_{t_0}^{(2)}(2, 2) \oplus Q_{t_0}^{(3)}(7:26, 7:26) \oplus Q_{t_0}^{(4)},$$

where $\oplus$ is the Kronecker sum operator. Due to the definition of Kronecker sums, each such Kronecker sum is a sum of four Kronecker products in which all but one of the matrices are identity. The non-identity matrix in each Kronecker product appears in the same position as in the Kronecker sum. That state changes do not take place in any but one of the LLM matrices with t_0 in each such Kronecker product explains why t_0 is called a local transition. Therefore, when t_0 appears in the off-diagonal of the HLM matrix, for instance, as in element (1,3) of the HLM matrix, the corresponding Kronecker product is

$$I_{15} \otimes Q_{t_0}^{(2)}(0:1, 2) \otimes I_{20} \otimes I_{30}.$$

See again the mapping of LLM states to HLM states in Table 1.

Q is a block matrix having as many blocks in each dimension as the number of macrostates (i.e., order of the HLM matrix). Each block is a sum of Kronecker products as dictated by the elements of the HLM matrix. For instance, element (4,4) of the HLM matrix is a sum of 2 Kronecker products (due to transitions t_1 and t_5) and 1 Kronecker sum (due to local transition t_0), thus effectively a sum 6 Kronecker products; whereas, element (6,1) consists of a single Kronecker product (due to transition t_2). The diagonal of Q is formed of the negated row sums of the HLM matrix (so that Q has row sums of zero) and may be stored explicitly or can be generated as needed. Although the CTMC in this example has 84,600 states and

410,160 nonzeros, the Kronecker representation associated with the HMM needs to store 1 HLM matrix having 29 nonzeros (i.e., count of the transitions in the HLM matrix) and 14 LLM matrices (since identity matrices are not stored) having a total of 281 nonzeros. Now let us turn to block partitionings induced by the Kronecker structure of the HMM.

In Table 2, we provide four nested partitionings along the diagonal of Q induced by the Kronecker structure of the HMM. The columns blks and order list respectively number and order of blocks in each macrostate for the given partitioning. We provide some more information under column cdts for levels 1–3 which is discussed in Section 3.1. Since the HLM has multiple macrostates, there exists a partitioning at level 0. The diagonal blocks at level 0 can be partitioned further as defined by LLM 1 at level 1 (i.e., one block is defined for each state of LLM 1). LLM 2 defines the next level of partitioning (i.e., one block is defined for each pair of states of LLM 1 and LLM 2), and LLM 3 (i.e., next to last LLM) defines the last nested partitioning. In HMMs, diagonal blocks at a given level of nested partitioning of Q are all square, but can have different orders in different macrostates. Consequently, off-diagonal blocks for the same level of nested partitioning in the off-diagonal part of the HLM matrix need not be square. This is different than SANs in which all (diagonal and off-diagonal) blocks at a level of nested partitioning associated with the Kronecker structure are square and have the same order.

For instance, a particular three-level BSOR iteration on this example can be one using the partitioning at level 0 with 7 diagonal blocks and employing BGS to solve each of these blocks using the nested 15 diagonal blocks at level 1. On the other hand, a usual (two-level) BSOR iteration can be one using the partitioning at level 1 having a total of 105 diagonal blocks. In this paper, we solve the diagonal blocks at the third level (or at the second level if it is a two-level BSOR iteration) directly as suggested in [16,23,24,35]. This is justified by the one time factorization of diagonal blocks, the relatively short time it takes, the sparsity of the factors, and the techniques we develop next.

Table 2
Four nested partitionings along the diagonal in *courier_small*

HLM	Level 0		Level 1			Level 2			Level 3		
state	blks	ordr	blks	cdts	ordr	blks	cdts	ordr	blks	cdts	ordr
0	1	22,500	15	6	1500	750	750	30	750	750	30
1	1	18,000	15	6	1200	30	30	600	600	450	30
2	1	450	15	15	30	15	15	30	15	15	30
3	1	9000	15	15	600	15	15	600	300	225	30
4	1	5400	15	6	360	45	45	120	180	135	30
5	1	5850	15	6	390	195	195	30	195	195	30
6	1	23,400	15	6	1560	195	195	120	780	585	30
$\sum =$	7		105	60		1245	1245		2820	2355	

3. Taking advantage of candidate blocks

The diagonal blocks that correspond to a partitioning of an irreducible CTMC have negative diagonal elements and nonnegative off-diagonal elements. Such diagonal blocks are known to be nonsingular [5]. When the CTMC results from an HMM, the diagonal blocks may have other properties as we next explain.

3.1. Detecting candidate blocks

First we remark that Kronecker sums contribute only to the diagonal of the HLM matrix. Furthermore, the diagonal blocks (at each level) of a Kronecker sum are all identical up to their diagonals and the diagonals differ by a multiple of the identity matrix. Therefore, in each macrostate in order to detect diagonal blocks with identical off-diagonal parts and diagonals differing from each other by a multiple of the identity matrix, we must check conditions related to Kronecker products only. Hereafter, we call the diagonal blocks satisfying the described property as *candidate blocks* in that macrostate.

Note that the set of diagonal blocks in a macrostate may form multiple partitions of candidate blocks, where blocks in each partition satisfy the definition of candidacy but two blocks in different partitions either have different off-diagonal parts or their diagonals do not differ from each other by a multiple of the identity matrix. Detecting all such partitions is a difficult process owing it to the various ways in which Kronecker products contribute to diagonal blocks. Hence, the following algorithm we present may not detect all candidate blocks. Nevertheless, we will be content with the ones it detects since it executes rapidly and we do not want to compute more than one real Schur factorization per macrostate.

Algorithm 1 can be used to check candidacy among all diagonal blocks in a given macrostate at a specific partitioning level. Together the two conditions C1 and C2 in the algorithm are sufficient to detect candidate blocks and they can be checked using the matrices that describe the HLM and the LLMs which are held in row sparse format. For instance, in macrostate 0 of the *courier_small* problem, the Kronecker products due transitions t_1 to macrostate 0, t_2 to macrostate 4, t_3 to macrostate 5, and t_2 to macrostate 6 must be checked (see the HLM matrix in Eq. (1)).

The first condition, C1, in Algorithm 1 ensures that the off-diagonal parts of all candidate blocks are identical and their diagonals differ from each other by a multiple of the identity matrix. In particular, we check parts C1(b) and C1(c) by inspecting the matrices of the LLMs respectively up to and after the partitioning level in the Kronecker product and making sure they are all identity matrices. The second condition, C2, ensures after the first condition that the contribution from off-diagonal blocks to the diagonals of candidate blocks do not alter the fact that the diagonals of candidate blocks differ from each other by a multiple of the identity matrix. In particular, we check parts C2(b) and C2(c) by inspecting the matrices of the LLMs respectively up to and after the partitioning level and making sure they have equal row sums.

Algorithm 1. Detecting candidate blocks in macrostate j at level l
For each Kronecker product that contributes to macrostate j:

C1. In diagonal blocks at level l in macrostate j, make sure there is:
   (a) either no contribution from the Kronecker product to the candidate block, or
   (b) the same contribution from the Kronecker product to all diagonal blocks (including the candidate block), or
   (c) a contribution from the Kronecker product that is a multiple of the identity matrix to some diagonal blocks (including the candidate block);

C2. In off-diagonal blocks at level l in macrostate j, make sure there is:
   (a) either no contribution from the Kronecker product to the off-diagonal blocks in the same row as the candidate block, or
   (b) the same contribution from the Kronecker product to the diagonals of all diagonal blocks (including the candidate block), or
   (c) a contribution from the Kronecker product that is a multiple of the identity matrix to the diagonals of some diagonal blocks (including the candidate block).

Example (continued). Table 2 lists the number of candidate blocks under the cdts columns in our running example for different levels of partitioning obtained by executing Algorithm 1. Note that we have excluded the partitioning at level 0 since it gives rise to one block in each macrostate. We remark that there is a relatively large number of candidate blocks in each partitioning. There is even a partitioning in which all diagonal blocks are candidates.

Next we explain how the candidate blocks in a macrostate can all take advantage of the real Schur factorization of one of the candidate blocks.

3.2. Using the real Schur factorization

First we recall that the real Schur factorization of a real nonsymmetric square matrix B exists [33, p. 114] and can be written as $B = ZTZ^{\mathrm{T}}$. This eigenvalue revealing factorization is characterized by the two matrices T and Z, and, although expensive to compute, has some useful properties. The matrix T is quasi-triangular meaning it is block triangular with blocks of order 1 or 2 along the diagonal; the blocks of order 1 contain the real eigenvalues of B and the blocks of order 2 contain the pairs of complex conjugate eigenvalues of B. On the other hand, the matrix Z is orthogonal (i.e., $Z^{\mathrm{T}}Z = I$) and contains the real Schur vectors. When both T and Z are requested, the cost of factorizing B of order m into real Schur form, assuming it is full, is $25m^3$ [19, p. 185]. Note that B can also be in the form $B = (ZP)(P^{\mathrm{T}}TP)(ZP)^{\mathrm{T}}$ for a permutation P in which $P^{\mathrm{T}}TP$ is quasi-triangular. Throughout this paper, we will assume without loss of generality that T is quasi-upper-triangular.

Now, let us assume that B_1 is the first candidate block in the macrostate under consideration. Let B_i, $i > 1$, represent the ith candidate block in the particular macrostate. Let

$$B_i - B_1 = \lambda_i I$$

and the real Schur form of B_1 be given by

$$B_1 = ZTZ^{\mathrm{T}}.$$

Then

$$B_i = Z(T + \lambda_i I)Z^{\mathrm{T}}.$$

Hence, if we are to solve

$$p_i B_i = b_i,$$

where p_i and b_i are row subvectors of appropriate length, then we can do so by solving the equivalent system

$$p_i Z(T + \lambda_i I)Z^{\mathrm{T}} = b_i.$$

This can be accomplished in three steps:

1. compute $c_i = b_i Z$;
2. let $y_i = p_i Z$; solve $y_i(T + \lambda_i I) = c_i$ for y_i;
3. compute $p_i = y_i Z^{\mathrm{T}}$;

and requires two vector-matrix multiplies and one quasi-triangular solve. All needs to be done is to store λ_i for each candidate block and the real Schur factors T and Z in each macrostate.

Example (continued). In its first two lines, Table 3 compares the total number of nonzeros in the LU factors L and U of all diagonal blocks (i.e., n/o, nc) with those of the case in which all candidate blocks in each macrostate use the common real Schur factors T and Z (i.e., n/o, ac). We call the former the *no candidates* (nc) case and the latter the *all candidates* (ac) case. The numbers in Table 3 and all such tables exclude the λ_is in the ac case and the nonzeros along the diagonals in all factors. It is interesting to note that the real Schur factors T and Z are quite sparse. This not only suggests a reduction in storage but also a possible reduction in solution time. In preliminary numerical experiments, we noticed that this is a property of candidate blocks having real eigenvalues (i.e., triangular T factor). For the real Schur factorization to be worthwhile the effort, it should not yield excessive number of nonzeros in the real Schur factors T and Z. On the other hand, we noticed that when the candidate block has complex eigenvalues, the T and Z factors are either completely or nearly completely full. Table 3 has some more information in its last two lines which will be discussed in Section 3.4.

Table 3
Nonzeros of factors of nc versus ac in *courier_small*

	Level 1		Level 2		Level 3	
	LU	Schur	LU	Schur	LU	Schur
n/o, nc	2,201,430	0	785,370	0	397,620	0
n/o, ac	1,245,411	32,154	0	8403	65,565	931
cmd, nc	649,320	0	365,640	0	208,680	0
cmd, ac	352,278	32,154	0	8403	34,410	931

Now let us state sufficient conditions for the existence of diagonal blocks with real eigenvalues in HMMs.

3.3. Diagonal blocks with real eigenvalues

The following discussion is based on the HMM description of CTMCs introduced in Section 2. Recall that each diagonal block of Q appears in one macrostate along the diagonal of the HLM matrix and each diagonal element of the HLM matrix is a sum of one Kronecker sum (due to LLM local transition matrices) and zero or more Kronecker products (due to LLM non-local transition matrices). The following lemma is used in stating Propositions 1 and 2 of this subsection.

Lemma 1. *The Kronecker product and the Kronecker sum of two nonsymmetric square matrices with real eigenvalues have real eigenvalues, and therefore are triangularizable using orthogonal matrices.*

Proof. Let the two nonsymmetric square matrices with real eigenvalues be X_1 and X_2. Then there must be orthogonal matrices, respectively, Z_1 and Z_2, that upper-triangularize the two matrices. In other words, we must have that $X_1 = Z_1 T_1 {Z_1}^{\mathrm{T}}$ and $X_2 = Z_2 T_2 {Z_2}^{\mathrm{T}}$, where T_1 and T_2 are the upper-triangular factors in the real Schur factorization of each matrix (see Section 3.2). Then

$$\begin{aligned} X_1 \otimes X_2 &= (Z_1 T_1 {Z_1}^{\mathrm{T}}) \otimes (Z_2 T_2 {Z_2}^{\mathrm{T}}) \\ &= (Z_1 \otimes Z_2)(T_1 \otimes T_2)(Z_1^{\mathrm{T}} \otimes Z_2^{\mathrm{T}}) \\ &= (Z_1 \otimes Z_2)(T_1 \otimes T_2)(Z_1 \otimes Z_2)^{\mathrm{T}} \end{aligned}$$

and

$$\begin{aligned} X_1 \oplus X_2 &= (Z_1 T_1 {Z_1}^{\mathrm{T}}) \otimes I + I \otimes (Z_2 T_2 {Z_2}^{\mathrm{T}}) \\ &= (Z_1 T_1 {Z_1}^{\mathrm{T}}) \otimes (Z_2 I {Z_2}^{\mathrm{T}}) + (Z_1 I {Z_1}^{\mathrm{T}}) \otimes (Z_2 T_2 {Z_2}^{\mathrm{T}}) \\ &= (Z_1 \otimes Z_2)(T_1 \otimes I)(Z_1^{\mathrm{T}} \otimes Z_2^{\mathrm{T}}) + (Z_1 \otimes Z_2)(I \otimes T_2)(Z_1^{\mathrm{T}} \otimes Z_2^{\mathrm{T}}) \\ &= (Z_1 \otimes Z_2)(T_1 \oplus T_2)(Z_1 \otimes Z_2)^{\mathrm{T}} \end{aligned}$$

from the compatibility of Kronecker product with ordinary matrix multiplication and ordinary matrix transposition. The matrix $(Z_1 \otimes Z_2)$ is orthogonal since Z_1 and Z_2 are orthogonal, and the matrices $(T_1 \otimes T_2)$ and $(T_1 \oplus T_2)$ are upper-triangular since T_1 and T_2 are upper-triangular [37]. $\square$

In order to state the two propositions in sufficient detail but without overcomplicating the notation, we need the following definition.

Definition 1. Let the diagonal block (j, j) of Q corresponding to element (j, j) of the HLM matrix be denoted by $Q_{j,j}$. Then

$$Q_{j,j} = \bigoplus_{k=1}^{K} Q_{t_0}^{(k)}(\mathscr{S}_j^{(k)}, \mathscr{S}_j^{(k)}) + \sum_{t_e \in \mathscr{T}_{j,j}} \mathrm{rate}_{t_e}(j, j) \bigotimes_{k=1}^{K} Q_{t_e}^{(k)}(\mathscr{S}_j^{(k)}, \mathscr{S}_j^{(k)}) + D_j,$$

where K is the number of LLMs, $S_j^{(k)}$ is the subset of states of LLM k mapped to macrostate j, $\mathscr{T}_{j,j}$ is the set of LLM non-local transitions in element (j, j) of the HLM matrix, $\mathrm{rate}_{t_e}(j, j)$ is the rate associated with transition $t_e \in \mathscr{T}_{j,j}$, and D_j is the diagonal (correction) matrix that sums the rows of Q corresponding to macrostate j to zero.

Furthermore, let the real Schur factorization of the local transition submatrix of LLM k in element (j, j) of the HLM matrix be given by

$$Q_{t_0}^{(k)}(\mathscr{S}_j^{(k)}, \mathscr{S}_j^{(k)}) = Z_k T_k {Z_k}^{\mathrm{T}},$$

where T_k is the (quasi-)upper-triangular factor and Z_k is the orthogonal factor.

We remark that $Q_{j,j}$ is a sum of three terms: the first due to the Kronecker sum of LLM local transition submatrices in element (j, j) of the HLM matrix, the second due to the sum of Kronecker products of LLM non-local transition submatrices in element (j, j) of the HLM matrix and the third due to the diagonal correction necessary to have row sums of zero in macrostate j of Q. Let $\tilde{D}_j$ denote the diagonal block of D_j associated with the particular diagonal block under consideration for real Schur factorization.

Proposition 1. *If for macrostate j:*

(a) $\mathscr{T}_{j,j} = \emptyset$, *and*
(b) *each T_k for $k > l$ is upper-triangular, and*
(c) $(\bigotimes_{k>l} {Z_k}^{\mathrm{T}})\tilde{D}_j(\bigotimes_{k>l} Z_k)$ *is diagonal,*

then the diagonal block under consideration at level l in macrostate j has real eigenvalues.

Proof. Without loss of generality, let $K = 2$ and $l = 0$. From Lemma 1 and Definition 1, we have

$$Q_{j,j} = (Z_1 \otimes Z_2)(T_1 \oplus T_2)(Z_1^{\mathrm{T}} \otimes Z_2^{\mathrm{T}}) + D_j.$$

Using the compatibility of Kronecker product with ordinary matrix multiplication, this can be written as

$$\begin{aligned} Q_{j,j} &= (Z_1 \otimes Z_2)(T_1 \oplus T_2)(Z_1^{\mathrm{T}} \otimes Z_2^{\mathrm{T}}) + (Z_1 \otimes Z_2)(Z_1^{\mathrm{T}} \otimes Z_2^{\mathrm{T}})D_j(Z_1 \otimes Z_2)(Z_1^{\mathrm{T}} \otimes Z_2^{\mathrm{T}}) \\ &= (Z_1 \otimes Z_2)(T_1 \oplus T_2 + (Z_1^{\mathrm{T}} \otimes Z_2^{\mathrm{T}})D_j(Z_1 \otimes Z_2))(Z_1^{\mathrm{T}} \otimes Z_2^{\mathrm{T}}). \end{aligned}$$

Note that since $Z_1 \otimes Z_2$ is orthogonal and D_j is diagonal, the matrix $(Z_1^{\mathrm{T}} \otimes Z_2^{\mathrm{T}})D_j(Z_1 \otimes Z_2)$ is symmetric. Part (b) of Proposition 1 ensures that $T_1 \oplus T_2$ is upper-triangular. Its part (c) together with part (b) ensures that $T_1 \oplus T_2 + (Z_1^{\mathrm{T}} \otimes Z_2^{\mathrm{T}})D_j(Z_1 \otimes Z_2)$ is upper-triangular. □

We remark that part (a) of Proposition 1 is satisfied by all macrostates along the diagonal of the HLM matrix in many HMMs arising from closed queueing networks. Its part (b) is satisfied, for instance, when the submatrices of the local transition matrices that are mapped to the particular macrostate for LLMs $(l + 1)$ and higher are triangular.

Note that Proposition 1 also describes an approach to construct the T and Z factors of the diagonal block that is to be real Schur factorized at level l in macrostate j from the real Schur factors of the LLM local transition submatrices and D_j. In fact,

$$Z = \bigotimes_{k>l} Z_k \quad \text{and} \quad T = \bigoplus_{k>l} T_k + \Big(\bigotimes_{k>l} {Z_k}^{\mathrm{T}}\Big)\widetilde{D}_j\Big(\bigotimes_{k>l} Z_k\Big).$$

We find it surprising that it is also possible to check part (c) in Proposition 1 and build the product using the orthogonal real Schur factors of LLM local submatrices as follows (we again consider the case with $K = 2$ and $l = 0$ for simplicity of notation):

$$\begin{aligned} &(Z_1^{\mathrm{T}} \otimes Z_2^{\mathrm{T}})D_j(Z_1 \otimes Z_2) \\ &\quad = (Z_1^{\mathrm{T}} \otimes Z_2^{\mathrm{T}})\Big(\sum_{s_1 \in \mathscr{S}_j^{(1)}} (e_{s_1}{e_{s_1}}^{\mathrm{T}}) \otimes D_j(s_1, s_1)\Big)(Z_1 \otimes Z_2) \qquad (2) \\ &\quad = \sum_{s_1 \in \mathscr{S}_j^{(1)}} ({Z_1}^{\mathrm{T}} e_{s_1}{e_{s_1}}^{\mathrm{T}} Z_1) \otimes ({Z_2}^{\mathrm{T}} D_j(s_1, s_1) Z_2). \qquad (3) \end{aligned}$$

Here, $s_1 \in \mathscr{S}_j^{(1)}$ and denotes a state of LLM 1. The submatrix $D_j(s_1, s_1)$, which has the same order as that of Z_2, is the diagonal block of D_j corresponding to state s_1 in LLM 1. Finally, e_{s_1} is the vector of zeros except a one in element s_1.

A few comments regarding expression (3) should be made. The matrix $({Z_1}^{\mathrm{T}} e_{s_1})(e_{s_1}{}^{\mathrm{T}} Z_1)$ is a symmetric outer-product and has rank-1 when ${Z_1}^{\mathrm{T}} e_{s_1} \neq 0$. Therefore, it is either diagonal or a (nondiagonal) symmetric matrix. If we encounter

a single such outer-product that is not a diagonal matrix, we may choose to conclude part (c) of Proposition 1 does not hold. When the outer-product $({Z_1}^{\mathrm{T}} e_{s_1})({e_{s_1}}^{\mathrm{T}} Z_1)$ is diagonal, together with the fact that it is of rank-1, this implies it must have a single nonzero along its diagonal. But this is possible if and only if Z_1 has one nonzero in row s_1. When all outer products in expression (3) are diagonal, it must be that Z_1 has one nonzero in each row, and these nonzeros are in different columns of Z_1 since it is orthogonal. This is possible, for instance, when Z_1 is the identity. Observe that ${Z_2}^{\mathrm{T}} D_j(s_1, s_1) Z_2$ is symmetric, and for part (c) of Proposition 1 to hold it must be diagonal as well. Therefore, Z_2 must also have a single nonzero in each row. In conclusion, it is possible to check part (c) of Proposition 1 by making sure each Z_k for $k > l$ has one nonzero per row. But this is equivalent to having each Z_k as a permutation matrix, and the Kronecker product of permutation matrices is a permutation matrix. We summarize this in the following corollary.

Corollary 1. *If Proposition* 1 *holds at level l in macrostate j and all Z_k for $k > l$ are permutation matrices, then the orthogonal real Schur factor of the diagonal block under consideration for real Schur factorization at that level is a permutation matrix and its upper-triangular factor has the same number of nonzeros as the diagonal block.*

In other words, real Schur factorization will amount to computing a permutation matrix that upper-triangularizes the particular diagonal block. A similar corollary holds for the next proposition, which subsumes Proposition 1 but is given separately; because, as we will see in the *courier* problem, there may be macrostates satisfying Proposition 1 and others satisfying Proposition 2.

Proposition 2. *If for macrostate j:*

(a) *each T_k for $k > l$ is upper-triangular, and*
(b) *each $\bigotimes_{k>l}({Z_k}^{\mathrm{T}} Q_{t_e}^{(k)}(\mathscr{S}_j^{(k)}, \mathscr{S}_j^{(k)}) Z_k)$ that contributes to the diagonal block under consideration at level l for all $e \in \mathscr{T}_{j,j}$ is upper-triangular, and*
(c) *$(\bigotimes_{k>l} {Z_k}^{\mathrm{T}}) \tilde{D}_j (\bigotimes_{k>l} Z_k)$ is diagonal,*

then the diagonal block under consideration at level l in macrostate j has real eigenvalues.

The proof of Proposition 2 is similar to that of Proposition 1, and it is possible to construct the real Schur factors of the particular diagonal block at level l in macrostate j from the real Schur factors of LLM local submatrices, the LLM nonlocal submatrices, and D_j. Checking part (b) of the proposition requires one to have previously computed the multipliers that multiply each Kronecker product in forming the candidate block when $l > 0$. However, this is something we do in detecting

candidate blocks and use in step C1(a) of Algorithm 1. In other words, those Kronecker products that will eventually get multiplied with a zero and therefore do not contribute to the diagonal block under consideration should not change our decision regarding the satisfiability of Proposition 2. Note also that it suffices for the first nondiagonal factor in the Kronecker product of part (b) to be an upper-triangular matrix to satisfy the condition for the particular $e \in \mathscr{T}_{j,j}$ (see Appendix A in [35, pp. 181–183]). Finally, it is possible to state a corollary similar to Corollary 1 for Proposition 2.

Example (continued). In *courier_small*, all macrostates except 2 satisfy Proposition 2 and macrostate 2 satisfies Proposition 1 (see the HLM matrix in Eq. (1)) for nested partitioning levels 1–3. In each macrostate, the Z factor is a permutation matrix and the T factor has as many nonzeros as in the candidate blocks. We emphasize that even though l may be small and diagonal blocks at that level large, the two propositions require the real Schur factorization of LLM local submatrices only. Even then, real Schur factorization needs to be employed only when the input matrix is not triangular. And when either of the two propositions is satisfied, we have a very efficient way of constructing the real Schur factors of candidate blocks.

Now we explain how the situation in LU factorized (non-candidate) diagonal blocks can be improved.

3.4. Applying COLAMD

During LU factorization of a sparse matrix, elements that are previously zero may become nonzero, a concept known as fill-in [34, p. 61]. Ordering the rows and/or columns of a sparse matrix can result in different numbers of nonzeros in the computed LU factors, and fill-reducing orderings for sparse matrices have been an active research area for the last thirty years. Our objective is to improve the fill-in generated by the LU factorization of (non-candidate) diagonal blocks.

The column approximate minimum degree (COLAMD) ordering algorithm [17, 18] computes a permutation vector such that the LU factorization of the column permuted matrix tends to be sparser than that of the unpermuted matrix. Once columns of the matrix are permuted, row permutation due to partial pivoting may take place during LU factorization to ensure stability. However, the row permutation used in partial pivoting is likely to be different than the column permutation returned by COLAMD resulting in a nonsymmetric permutation.

In block partitionings of irreducible CTMCs, the transpose of each diagonal block is column diagonally dominant. This suggests transposing each (non-candidate) diagonal block, column permuting the transposed block according to COLAMD, and using the column permutation returned by COLAMD as the row permutation in partial pivoting during its LU factorization. This has the additional advantage that all multipliers are bounded by one during LU factorization. Hence, the transposed (non-

candidate) diagonal blocks can be symmetrically permuted according to COLAMD and then LU factorized. However, this requires that the right-hand side is permuted before carrying out forward–backward substitutions and the resulting solution vector is inverse permuted.

Example (continued). In Table 3, we have also included the number of nonzeros obtained in the LU factors of our running example when COLAMD is used. Compare the rows of n/o (i.e., natural or original ordering) with those of cmd (i.e., COLAMD); the results are encouraging.

In Section 4, we discuss the particular three-level BSOR implementation and comment on implementation issues.

4. Implementation considerations

Algorithm 2 that is presented next serves as set-up for the BSOR solver and factorizes diagonal blocks. Although not shown, it also performs some preprocessing to determine the size of the arrays to be allocated. For each macrostate j, it sets the pair of nested three-level partitioning parameters, $(l_1(j), l_2(j))$ and then factorizes the diagonal blocks at level $l_2(j)$ based on certain parameters.

Algorithm 2. Factorization of diagonal blocks in HMMs.
For each macrostate j:

1. Set $l_1(j)$ and $l_2(j)$:
   (a) If (# of microstates in macrostate $j \leqslant MAX_LU_ORDER$), $l_1(j) = l_2(j) = 0$;
   (b) Else if using fixed level partitioning, $l_1(j) = LEVEL$, else compute lowest $l_1(j)$ at which (order of diagonal blocks in macrostate $j \leqslant MAX_BLOCK_ORDER$); $l_2(j) = l_1(j) + OFFSET$;
2. Factorize diagonal blocks at level $l_2(j)$:
   (a) Detect candidate blocks at level $l_2(j)$ in macrostate j using Algorithm 1;
   (b) If ((macrostate j at level $l_2(j)$ does not satisfy Propositions 1 and 2) and (order of diagonal blocks in macrostate $j > MAX_SCHUR_ORDER$)) or (# of candidate blocks in macrostate $j \leqslant MIN_SCHUR$), use nc factorization;
   (c) Else use ac factorization.

Macrostates having a small number of microstates can be LU factorized directly at level 0 (see the parameter MAX_LU_ORDER in step 1(a)). In other macrostates, the parameter $l_1(j)$ can be set either by using the fixed level of partitioning specified by the parameter $LEVEL$, or adaptively by using the parameter

MAX_BLOCK_ORDER (see step 1(b)). Note that it is possible to use different pairs of $(l_1(j), l_2(j))$ values in different macrostates. When the HLM has a single macrostate, $l_1(0) > 0$ should hold, otherwise an LU factorization of Q may be initiated. Once $l_1(j)$ is set, the parameter $l_2(j)$ is computed by adding the parameter $OFFSET$ to $l_1(j)$.

Assuming that the HLM is numbered 0, $l_1(j) < l_2(j)$, and at least one LLM among the LLMs $(l_1(j) + 1)$ through $l_2(j)$ has multiple states mapped to macrostate j, in macrostate j we have a nested three-level partitioning of which the first level lies between the HLM and the model with index $l_1(j)$, the second level lies between LLM $(l_1(j) + 1)$ and LLM $l_2(j)$, and the third level lies between LLM $(l_2(j) + 1)$ and the last LLM. The diagonal blocks that get factorized appear at the third level and are defined by $l_2(j)$. BGS is used to solve the diagonal blocks at the second level, which are defined by $l_1(j)$ and $l_2(j)$. The outer BSOR iteration takes place at the first level and is defined by $l_1(j)$. If $l_1(j) = l_2(j)$, we have the usual (two-level) BSOR iteration over macrostate j.

For each macrostate, candidate blocks are determined using Algorithm 1 in Section 3.1. If the number of candidate blocks is greater than MIN_SCHUR and either Propositions 1 and 2 are satisfied or there is sufficient space to carry out a non-sparse real Schur factorization, the real Schur factorization of the first candidate block in that macrostate is performed. This can be done in two different ways.

When the particular macrostate satisfies Propositions 1 or 2, the real Schur factors of the first candidate block are constructed as described in Section 3.3 from the LLM submatrices and their real Schur factors obtained using the CLAPACK routine `dgees` [19, p. 185] available at [29] (see step 2(c) in Algorithm 2). We call this the *sparse* approach. This routine effectively uses two two-dimensional double precision arrays the first of which has the particular matrix on input and the (quasi-)upper-triangular factor on output, whereas the second has the orthogonal factor of real Schur vectors on output. The returned real Schur factors are compacted and stored as sparse matrices to be used in the iterative part of the BSOR solver. In the sparse approach, we generate and store the real Schur factors of the candidate block in sparse format using the LLM submatrices and their sparse real Schur factors as discussed in Section 3.3.

When Propositions 1 and 2 are not satisfied, one can still compute the real Schur factors of the particular candidate block using `dgees`. We call this the *full* approach. The figures in Table 3 are obtained using this approach. We remark that the real Schur factors computed using the sparse approach and the full approach can be different, but will be related to each other by a permutation matrix, P, as mentioned at the beginning of Section 3.2. Nevertheless, their products always give the same diagonal block. Obviously, `dgees` limits the order of candidate blocks that can be (full) real Schur factorized on a given architecture. For instance, assuming that a double precision number is stored in eight bytes, candidate blocks with a maximum order of 3500 would require roughly 200 MB storage for the two two-dimensional arrays in the full approach. Note that this is temporary storage and not used in the iterative part

of the BSOR solver (i.e., steps 3–6 of Algorithm 3). This limitation is imposed with the parameter MAX_SCHUR_ORDER (see step 2(b)). We provide a comparison of the sparse and full approaches with regards to the time they take in the section on numerical results wherever possible.

While processing other candidate blocks in the same macrostate, the difference between the first diagonal element of each candidate block and that of the first candidate block (i.e, λ_i's in Section 3.2) must be stored so that later they can be used in the solution process. To expedite this process we choose rather to store the reciprocals of the diagonals of the matrices $(T + \lambda_i I)$ in each macrostate. This exchanges the addition and division per diagonal element at each real Schur solve with a one time division and a per solve multiplication per diagonal element. The non-candidate blocks all get sparse LU factorized. Before the factorizations are performed, COLAMD is run on the transpose of the non-candidate blocks after which they are symmetrically permuted. The COLAMD routine is available at [15]. When sufficient candidate blocks in a given macrostate do not exist, all diagonal blocks get sparse LU factorized using the COLAMD ordering. The parameter MIN_SCHUR can be used to find a better mix of real Schur and LU factorizations among macrostates considering the relatively long time to perform a real Schur factorization compared to an LU factorization when Propositions 1 and 2 are not satisfied.

Algorithm 3. Three-level BSOR solver for HMMs

1. Execute Algorithm 2 and factorize diagonal blocks;
2. Set current solution vector π by initial approximation, $it_1 = 1$;
3. If (relaxation parameter $w \neq 1$), copy π to previous solution vector π_{prev};
4. For each macrostate j, sequentially over the HLM matrix:

   (a) Compute negated right-hand side b:
   - Set $b = 0$; add to b product of π with blocks above and below element (j, j);
   - Add to b product of π with block strictly lower-triangular part at level $l_1(j)$ of element (j, j);

   (b) Solve block upper-triangular part at level $l_1(j)$ of element (j, j) for subvector j of π:
   - For each diagonal block j_1 at level $l_1(j)$ of element (j, j), sequentially:
     i. Solve block upper-triangular part at level $l_2(j)$ of diagonal block j_1 at level $l_1(j)$ in element (j, j) for subvector j_1 of π:
     A. Set b_{temp} and π_{temp} respectively by subvectors j_1 of b and π, $it_2 = 1$;
     B. Add to b product of π with block strictly lower-triangular part at level $l_2(j)$ of diagonal block j_1 at level $l_1(j)$ in element (j, j);
     C. For each diagonal block j_2 at level $l_2(j)$ of diagonal block j_1 at level $l_1(j)$ in element (j, j), sequentially:

– Solve diagonal block j_2 at level $l_2(j)$ of diagonal block j_1 at level $l_1(j)$ in element (j, j) with precomputed factors using negated subvector of b as right-hand side;
– Add to b product of subvector j_2 of π with corresponding blocks in block upper-triangular part at level $l_2(j)$ of diagonal block j_1 at level $l_1(j)$ in element (j, j);

D. If $(it_2 \geq MAX_IT_2)$ or (norm of difference between π_{temp} and subvector j_1 of $\pi \leqslant STOP_TOL_2$), then subvector j_1 of π has approximation at it_2; else set subvector j_1 of b by b_{temp}, π_{temp} by subvector j_1 of π, $it_2 = it_2 + 1$ and go to step 4(b)iB;

ii. If $(w \neq 1)$, set subvector j_1 of π by w times subvector j_1 of π plus $(1 - w)$ times subvector j_1 of π_{prev};

iii. Add to b product of subvector j_1 of π with corresponding blocks in block upper-triangular part at level $l_1(j)$ of element (j, j);

5. If $(it_1 \bmod NORM_COUNT == 0)$, normalize π;
6. If $(it_1 \geqslant MAX_IT_1)$ or (time $\geqslant MAX_TIME$) or (norm of residual vector $\leqslant STOP_TOL_1$), then stop, normalize π and take it as the steady state vector of the HMM; else set $it_1 = it_1 + 1$ and go to step 3.

In Algorithm 3, we present a high level description of the three-level BSOR solver for HMMs which aims at solving the singular linear system $\pi Q = 0$ subject to the normalization condition $\|\pi\|_1 = 1$, where π is the (row) stationary probability vector of Q. We assume that Q is irreducible; hence, the stationary vector of Q is also its steady state vector.

Steps 3–6 correspond to the outer BSOR iteration at the first level with relaxation parameter w. The BSOR iteration continues until user time reaches MAX_TIME, the number of iterations, it_1, reaches MAX_IT_1, or norm of the residual vector meets the prespecified tolerance, $STOP_TOL_1$. The residual vector is computed by premultiplying Q with the current solution vector, π, using the efficient vector-Kronecker product multiplication algorithm [21]. Note that the relaxation takes place in step 4(b)ii and step 5 ensures that π is normalized every $NORM_COUNT$ iterations.

Steps 4(b)iB–D correspond to the BGS iteration used to solve each diagonal block at the second level. Therein, the iteration continues until the number of iterations, it_2, reaches MAX_IT_2 or norm of the difference between successive approximations meets the prespecified tolerance, $STOP_TOL_2$.

Finally, the first part in step 4(b)iC corresponds to the direct solution of diagonal blocks at the third level by either performing forward–backward substitutions with LU factors or using the process described in Section 3.2 with real Schur factors. When $l_1(j) = l_2(j)$, we have a two-level solver and the BGS iteration at the second level becomes the direct solution of block j_1 in element (j, j) of the HLM matrix. Throughout Algorithm 3, the negated right-hand side b is used since the vector-Kronecker product multiplication routine is coded so as to add onto an input vector.

Therefore, right before solving a diagonal block at the third level, the appropriate subvector of b is negated and used as the right-hand side.

The iterative part of the three-level BSOR solver requires a maximum of three double precision arrays (i.e., π, π_{prev}, and an array to hold the diagonal of Q) each as long as the state space size, and four double precision work arrays each as long as the maximum number of microstates in a macrostate. Note that π_{prev} is allocated only if $w \neq 1.0$.

Section 5 discusses the results of numerical experiments with the three-level BSOR solver we name STR_BSOR.

5. Numerical results

We implemented the BSOR solver as discussed in Section 4 in C as part of the APNN toolbox [4]. All experiments are performed on a 550 MHz Pentium III processor and a 256 MB main memory under Linux. All times are reported as seconds of CPU time. In the tables, we report the times spent in Steps 1–2 and Steps 3–6 of Algorithm 3 respectively under columns S1-2 and S3-6, and indicate the fastest solvers in bold.

BSOR is known to be a competitive solver even when compared with IAD and ILU preconditioned projection methods [16]. In this paper, we have attempted to remedy its storage and time disadvantages related to the factorization of diagonal blocks in large systems and have provided a three-level version for HMMs with multiple macrostates. As opposed to sparse MCs, there are only a limited number of steady state solvers available for MCs underlying Kronecker representations. Therefore, we experiment with BSOR using different combinations of its parameters, and compare it with commonly used solvers for Kronecker representations.

We set $MAX_SCHUR_ORDER = 3500$, $MAX_IT_1 = 2000$, $MAX_TIME = 5000$ s, $STOP_TOL_1 = 10^{-8}$, $MAX_IT_2 = 10$, $STOP_TOL_2 = 10^{-3}$, $NORM_COUNT = 10$, and $w = 1.0$ (i.e., BGS outer iteration). MIN_SCHUR is set to zero when we experiment with the ac case and set to a large value when we experiment with the nc case. We have turned off the feature in Step 1(a) of Algorithm 2 and have used fixed levels of partitionings. Hence, the pair of parameters $(l_1(j), l_2(j))$ is independent of macrostate number, j, in the tables.

We consider two problems with multiple macrostates that appear in the literature and have been used as benchmarks. We compare all results with those of other HMM solvers available in the APNN toolbox. In particular, we compare STR_BSOR with STR_SOR, STR_RSOR, and STR_BICGSTAB. The STR_SOR solver implements a BSOR like method which uses diagonal blocks at level 0 with relaxation parameter w but does not attempt to solve the diagonal blocks. The solver STR_RSOR implements a point SOR method similar to the one discussed in [35]. The STR_BICGSTAB solver implements BiCGStab [36] as discussed in [3, pp. 27–28].

5.1. Courier protocol

We consider two HMMs of the Courier protocol introduced earlier in this paper. Both HMMs have 1 HLM and 4 LLMs; the LLMs are represented by 14 matrices. The underlying CTMCs of both HMMs are irreducible.

5.1.1. courier_small

The HMM named *courier_small* is discussed in Section 2. Its nested block partitionings, number of candidate blocks, and nonzeros in the factorized blocks are given in Tables 2 and 3. In Table 4, we present the results of STR_BSOR using a combination of different nested partitionings. Note that there is no advantage in using the COLAMD ordering with the ac case when $l_2 = 2$ since all diagonal blocks in this partitioning are already candidates. Furthermore, the extra time spent in S1-2 is offset by the decrease in S3-6 with COLAMD in the nc case. It can be said that the advantage of using COLAMD in this problem is limited to less fill-in.

The fastest solver utilizes the $(l_1, l_2) = (1, 2)$ partitioning with the sparse ac case and converges in 50 iterations and a total of 14 s. Note that the number of outer iterations do not change among the three partitionings employed. Although the time to execute S1-2 in the sparse ac case is smaller than that of the full ac case, this cannot be seen in Table 4. However, this discrepancy results due to rounding in the total solution time being slightly smaller for the winning solver. We remark that when $l_2 = 2$ the extra storage taken by the reciprocated diagonals of the matrices $(T + \lambda_i I)$ is as long as the state space size. This information although not available in Table 3 can be inferred from Table 2. The storage advantage of the sparse approach over the full approach is obvious, and its advantage in S1-2 will also become clear once we consider larger problems.

For the same problem, STR_SOR converges in 1500 iterations and 209 s, STR_RSOR converges in 490 iterations and 114 s, and STR_BICGSTAB converges in 268 iterations and 48 s. Note that number of iterations to convergence drops roughly to one tenth of that of STR_RSOR by using a STR_BSOR solver with $l_2 = 2$. Hence, we can say that STR_BSOR is a very effective solver for this problem. We have also considered STR_BSOR with the pair of parameters $(MAX_IT_2, STOP_TOL_2) = (30, 10^{-5})$. The number of outer iterations, it_1, decreased to 40 in the partitioning $(l_1, l_2) = (0, 2)$; however, the solution time did not change. Nevertheless, in some

Table 4
Performance of STR_BSOR in *courier_small*

		n/o, nc		cmd, nc		full, ac		sparse, ac	
(l_1, l_2)	it_1	S1-2	S3-6	S1-2	S3-6	S1-2	S3-6	S1-2	S3-6
(2,2)	50	2	17	3	16	1	15	**1**	14
(1,2)	50		16		15		14		**13**
(0,2)	50		19		19		18		17

cases we can expect to be better off by employing a stronger stopping tolerance in the BGS iteration at the second level when $l_1 - l_2 > 1$.

5.1.2. courier_medium

The HMM named *courier_medium* has an HLM matrix of order 10 with 47 nonzeros. The 4 LLMs respectively have 15, 217, 88, 30 states with partitioned state spaces in LLM 2 and LLM 3, and require a total of 845 nonzeros in the 14 LLM matrices. Q has 419,400 states and 2,281,620 nonzeros. Its nested block partitionings are similar to those of *courier_small*. Its partitioning at level 3 yields 13,980 blocks of order 30 out of which 11,265 are candidates; that at level 2 yields 4245 blocks of orders between 30 and 1800 all of which are candidates; that at level 1 yields 150 blocks of orders between 30 and 7800 of which 78 are candidates; and that at level 0 yields 10 blocks of orders between 450 and 117,000. As in *courier_small*, all macrostates in this problem except 2 satisfy Proposition 2 and macrostate 2 satisfies Proposition 1. Table 5 provides the numbers of nonzeros in the nc and ac cases with and without COLAMD ordering. Again, there is no advantage in using the COLAMD ordering with the ac case when $l_2 = 2$.

We experimented with the partitioning $l_2 = 2$ in which the extra storage taken by the reciprocated diagonals of its ac case is 419,400 nonzeros. Even though this number does not appear in Table 5, the number of nonzeros in the factors of the ac case is relatively small compared to that of the nc case. In Table 6 we provide the results with the STR_BSOR solver; the fastest solver utilizes the $(l_1, l_2) = (1, 2)$ partitioning with the sparse ac case and converges in 50 iterations and a total of 128 s. Note that in this problem there is a decrease in the number of outer iterations when a three-level partitioning is employed. Furthermore, the time spent in S1-2 of

Table 5
Nonzeros of factors of nc versus ac in *courier_medium*

	Level 2		Level 3	
	LU	Schur	LU	Schur
n/o, nc	4,763,820	0	1,971,180	0
n/o, ac	0	30,436	382,815	1330
cmd, nc	2,544,225	0	1,034,520	0
cmd, ac	0	30,436	200,910	1330

Table 6
Performance of STR_BSOR in *courier_medium*

		n/o, nc		cmd, nc		full, ac		sparse, ac	
(l_1, l_2)	it_1	S1-2	S3-6	S1-2	S3-6	S1-2	S3-6	S1-2	S3-6
(2,2)	60	19	180	34	170	12	161	**4**	160
(1,2)	50		140		132		124		**124**
(0,2)	50		182		173		164		164

the sparse approach is 8 s shorter than that of the full approach. The discrepancy between columns S3-6 of the full ac case and the sparse ac case is due to rounding as in *courier_small*. On the other hand, the 15 s difference in S1-2 between n/o and COLAMD orderings of the nc case is not offset in any of the STR_BSOR solvers. Therefore, as in *courier_small*, the advantage of using COLAMD is limited to less fill-in. In *courier_medium*, STR_SOR converges in 1190 iterations and 1293 s, STR_RSOR converges in 360 iterations and 551 s, and STR_BICGSTAB converges in 339 iterations and 399 s. STR_BSOR is a highly effective solver for this problem as well.

5.2. Multiserver multiqueue

We consider two HMMs of the multiserver multiqueue discussed in [1]. Both HMMs have 1 HLM, 5 LLMs, and six transitions denoted by t_0 through t_5 that take place in the HLM and affect the LLMs. Each LLM contributes to two transitions other than the local transition, t_0; hence, the LLMs are represented by 15 matrices. Though, none of the non-local transitions appear along the diagonal of the HLM matrix, and it turns out that for all macrostates in both *msmq* problems Corollary 1 applies at all partitioning levels. However, real Schur factors of the first candidate block computed using the sparse approach and the full approach are different and related through a permutation matrix for some macrostates. The underlying CTMCs are irreducible.

5.2.1. msmq_medium

The HMM named *msmq_medium* has an HLM matrix of order 15 with 25 nonzeros. The 5 LLMs all have 32 states with partitioned state spaces and require a total of 370 nonzeros in the 15 LLM matrices. Q has 358,560 states and 2,135,160 nonzeros. The results of this example are provided in Tables 7–9. The level 4 partitioning is omitted from Table 7 since it has very small blocks with orders between 6 and 15; the level 0 partitioning is omitted because there is only one block in each macrostate. Interestingly, all diagonal blocks are candidates in all levels of partitioning, and the extra storage for the reciprocated diagonals is 358,560 nonzero elements. Note that the number of nonzeros in the factors of the ac case in Table 8 is very small compared to that of the nc case for both levels of partitioning.

The fastest STR_BSOR solver utilizes the $(l_1, l_2) = (1, 2)$ partitioning with the sparse ac case and converges in 120 iterations and a total of 66 s (see Table 9). This is 3 s better than the next fastest solver. The high sparsity of the real Schur factors enables the ac case to converge much faster than the nc case in this problem. Note also that in *msmq_medium* the COLAMD ordering in the nc case yields smaller solution time than that of the n/o ordering. For the same problem, STR_SOR converges in 360 iterations and 131 s, STR_RSOR converges in 240 iterations and 112 s, STR_BICGSTAB converges in 325 iterations and 168 s. We remark that *msmq_medium* is a relatively easy problem compared to *courier_medium* judging by the time it takes

Table 7
Three nested partitionings along the diagonal in *msmq_medium*

HLM	Level 1			Level 2			Level 3		
state	blks	cdts	ordr	blks	cdts	ordr	blks	cdts	ordr
0	15	15	1296	90	90	216	540	540	36
1	11	11	2376	121	121	216	726	726	36
2	11	11	2376	66	66	396	726	726	36
3	11	11	2376	66	66	396	396	396	66
4	11	11	2376	66	66	396	396	396	66
5	6	6	3240	90	90	216	540	540	36
6	6	6	4356	66	66	396	726	726	36
7	6	6	4356	66	66	396	396	396	66
8	6	6	4356	66	66	396	396	396	66
9	6	6	3240	36	36	540	540	540	36
10	6	6	4356	36	36	726	396	396	66
11	6	6	4356	36	36	726	396	396	66
12	6	6	3240	36	36	540	216	216	90
13	6	6	4356	36	36	726	216	216	121
14	6	6	3240	36	36	540	216	216	90
$\sum =$	119	119		913	913		6822	6822	

Table 8
Nonzeros of factors of nc versus ac in *msmq_medium*

	Level 2		Level 3	
	LU	Schur	LU	Schur
n/o, nc	5,178,888	0	1,890,432	0
n/o, ac	0	40,634	0	4408
cmd, nc	3,213,042	0	1,326,276	0
cmd, ac	0	40,634	0	4408

Table 9
Performance of STR_BSOR in *msmq_medium*

		n/o, nc		cmd, nc		full, ac		sparse, ac	
(l_1, l_2)	it_1	S1-2	S3-6	S1-2	S3-6	S1-2	S3-6	S1-2	S3-6
(2,2)	120	12	109	22	91	7	67	**2**	67
(1,2)	120		106		89		64		**64**
(0,2)	120		111		94		68		68

to be solved by the basic solvers in the APNN toolbox. Even then the results with the STR_BSOR solver are quite good.

5.2.2. msmq_large

The HMM named *msmq_large* has an HLM matrix of order 35 with 75 nonzeros. The 5 LLMs all have 60 states with partitioned state spaces and require a total of 790 nonzeros in the 15 LLM matrices. Q has 2,945,880 states and 19,894,875 nonzeros. Its nested block partitionings are similar to those of *msmq_medium*. Its partitioning at level 4 yields 323,911 blocks of orders between 7 and 22 out of which 154,910 are candidates; that at level 3 yields 34,754 blocks of orders between 49 to 234 all of which are candidates; that at level 2 yields 3621 blocks of orders between 343 and 2197 all of which are candidates; that at level 1 yields 364 blocks of orders between 2401 and 15,379 all of which are candidates; and that at level 0 yields 35 blocks of orders between 52,822 and 107,653.

We experimented with the partitioning $l_2 = 2$ in which the extra storage taken by the reciprocated diagonals of its ac case is 2,945,880. Even though this number does not appear in Table 10, the number of nonzeros in the factors of the ac case is still quite small compared to that of the nc case. Due to high fill-in (see Table 10), it was not possible to experiment with the nc cases in this problem. The results of experiments with the ac cases are available in Table 11. The fastest STR_BSOR solver uses $(l_1, l_2) = (1, 2)$ with the sparse ac case and converges in 120 iterations and a total of 807 s (of which 26 s are spent in S1-2). Compare the 26 s spent in S1-2 of the sparse ac case with the 272 s spent in S1-2 of the full ac case. The cubic time complexity of the real Schur factorization manifests itself clearly in *msmq_large*. The discrepancy in S3-6 between the sparse ac case and the full ac case may be due to having different real Schur factors for the first candidate block in some macrostates as indicated at the beginning of Section 5.2. For the same problem, STR_SOR converges in 360

Table 10
Nonzeros of factors of nc versus ac in *msmq_large*

	Level 2		Level 3	
	LU	Schur	LU	Schur
n/o, nc	15,739,038	0	6,903,785	0
n/o, ac	0	214,533	0	17,868
cmd, nc	13,337,267		6,133,897	
cmd, ac	0	214,533	0	17,868

Table 11
Performance of the BSOR solver in *msmq_large*

		full, ac		sparse, ac	
(l_1, l_2)	it_1	S1-2	S3-6	S1-2	S3-6
(2,2)	120	272	819	**26**	817
(1,2)	120		782		**781**
(0,2)	120		843		842

iterations and 1632 s, STR_RSOR converges in 190 iterations and 977 s, and STR_BICGSTAB converges in 401 iterations and 2221 s. 190 iterations with STR_RSOR versus the 120 iterations with STR_BSOR in this problem justifies using STR_BSOR only with the sparse ac case.

6. Conclusion

It is shown that MCs in the form of sums of Kronecker products have considerable structure that may be exploited in iterative solution methods. A three-level BSOR solver that exploits this inherent structure is presented for HMMs. In almost all experiments, there happens to be a three-level BSOR solver which converges faster than the usual (two-level) BSOR solver. For the faster converging three-level BSOR solvers, the nested partitioning levels are found to be adjacent. The idea of using one real Schur factorization for diagonal blocks that differ from each other by a multiple of the identity (that is, the so-called candidate blocks) tends to reduce both storage and time taken by the BSOR solver. There are problems in which it is possible to construct the real Schur factors from the real Schur factors of component matrices. Finally, COLAMD produces good fill-reducing orderings and can be used in solving nonsingular linear systems associated with MC problems.

The three-level BSOR solver may be improved in a variety of ways one of which is to consider locating other candidate blocks not detected by Algorithm 1. A second direction may be to reorder LLMs so that larger and/or more candidate blocks that satisfy Proposition 1 or 2 appear along the diagonal of the underlying MC. But more importantly, the relatively small number of iterations associated with the BSOR solver to reach a prespecified tolerance and the storage required to implement it (especially when Propositions 1 and 2 are satisfied) suggest that it can be an effective preconditioner for HMMs.

Acknowledgements

This work has been carried out at Dresden University of Technology, where the second author was a research fellow of the Alexander von Humboldt Foundation.

References

[1] M. Ajmone-Marsan, S. Donatelli, F. Neri, GSPN models of Markovian multiserver multiqueue systems, Performance Evaluation 11 (1990) 227–240.
[2] APNN-Toolbox case studies home page at http://www4.cs.uni-dortmund.de/APNN-TOOLBOX/case_studies/.
[3] R. Barret, M. Berry, T. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, H. van der Vorst, Templates for the Solution of Linear Systems, SIAM Press, Philadelphia, Pennslyvania, 1994.

[4] F. Bause, P. Buchholz, P. Kemper, A toolbox for functional and quantitative analysis of DEDS, in: R. Puigjaner, N.N. Savino, B. Serra (Eds.), Quantitative Evaluation of Computing and Communication Systems, Lecture Notes in Computer Science 1469, Springer Verlag, 1998, pp. 356–359.
[5] A. Berman, R.J. Plemmons, Nonnegative Matrices in the Mathematical Sciences, SIAM Press, Philadelphia, Pennslyvania, 1994.
[6] P. Buchholz, A class of hierarchical queueing networks and their analysis, Queueing Systems 15 (1994) 59–80.
[7] P. Buchholz, Hierarchical structuring of superposed GSPNs, IEEE Transactions on Software Engineering 95 (1999) 166–181.
[8] P. Buchholz, Structured analysis approaches for large Markov chains, Applied Numerical Mathematics 31 (1999) 375–404.
[9] P. Buchholz, Projection methods for the analysis of stochastic automata networks, in: B. Plateau, W.J. Stewart, M. Silva (Eds.), Numerical Solution of Markov Chains, Prensas Universitarias de Zaragoza, Zaragoza, Spain, 1999, pp. 149–168.
[10] P. Buchholz, P. Kemper, On generating a hierarchy for GSPN analysis, Performance Evaluation Review 26 (1998) 5–14.
[11] P. Buchholz, G. Ciardo, S. Donatelli, P. Kemper, Complexity of memory-efficient Kronecker operations with applications to the solution of Markov models, INFORMS Journal on Computing 12 (2000) 203–222.
[12] P. Buchholz, M. Fisher, P. Kemper, Distributed steady state analysis of stochastic automata networks, in: B. Plateau, W.J. Stewart, M. Silva (Eds.), Numerical Solution of Markov Chains, Prensas Universitarias de Zaragoza, Zaragoza, Spain, 1999, pp. 76–95.
[13] J. Campos, S. Donatelli, M. Silva, Structured solution of asynchronously communicating stochastic modules, IEEE Transactions on Software Engineering 25 (1999) 147–165.
[14] G. Ciardo, A.S. Miner, A data structure for the efficient Kronecker solution of GSPNs, in: P. Buchholz, M. Silva (Eds.), Proceedings of the 8th International Workshop on Petri Nets and Performance Models, IEEE-CS Press, 1999, pp. 22–31.
[15] COLAMD home page at http://www.cise.ufl.edu/research/sparse/colamd/.
[16] T. Dayar, W.J. Stewart, Comparison of partitioning techniques for two-level iterative solvers on large, sparse Markov chains, SIAM Journal on Scientific Computing 21 (2000) 1691–1705.
[17] T.A. Davis, J.R. Gilbert, S.I. Larimore, E.G. Ng, A column approximate minimum degree ordering algorithm, Technical Report TR-00-005, Computer and Information Sciences Department, University of Florida, Gainesville, FL, 2000.
[18] T.A. Davis, J.R. Gilbert, S.I. Larimore, E.G. Ng, Algorithm 8xx: COLAMD, a column approximate minimum degree ordering algorithm, Technical Report TR-00-006, Computer and Information Sciences Department, University of Florida, Gainesville, FL, 2000.
[19] J.W. Demmel, Applied Numerical Linear Algebra, SIAM Press, Philadelphia, Pennslyvania, 1997.
[20] S. Donatelli, Superposed stochastic automata: a class of stochastic Petri nets with parallel solution and distributed state space, Performance Evaluation 18 (1993) 21–26.
[21] P. Fernandes, B. Plateau, W.J. Stewart, Efficient descriptor-vector multiplications in stochastic automata networks, Journal of the ACM 45 (1998) 381–414.
[22] P. Fernandes, B. Plateau, W.J. Stewart, Optimizing tensor product computations in stochastic automata networks, RAIRO Operations Research 32 (1998) 325–351.
[23] O. Gusak, T. Dayar, Iterative aggregation–disaggregation versus block Gauss–Seidel on continuous-time stochastic automata networks with unfavorable partitionings, in: M.S. Obaidat, F. Davoli (Eds.), Proceedings of the 2001 International Symposium on Performance Evaluation of Computer and Telecommunication Systems, Orlando, Florida, 2001, pp. 617–623.
[24] O. Gusak, T. Dayar, J.-M. Fourneau, Lumpable continuous-time stochastic automata networks, European Journal of Operational Research 148 (2003) 436–451.
[25] S. Haddad, P. Moreaux, Asynchronous composition of high-level Petri nets: a quantitative approach, in: J. Billington, W. Reisig (Eds.), Proceedings of the 17th International Conference on Application

and Theory of Petri Nets, Lecture Notes in Computer Science 1091, Springer Verlag, 1996, pp. 192–211.

[26] J. Hillston, L. Kloul, An efficient Kronecker representation for PEPA models, in: L. de Alfaro, S. Gilmore (Eds.), Proceedings of the 1st Process Algebras and Performance Modeling, Probabilistic Methods in Verification Workshop, Lecture Notes in Computer Science 2165, Springer Verlag, 2001, pp. 120–135.

[27] P. Kemper, Numerical analysis of superposed GSPNs, IEEE Transactions on Software Engineering 22 (1996) 615–628.

[28] V. Migallón, J. Penadés, D.B. Szyld, Block two-stage methods for singular systems and Markov chains, Numerical Linear Algebra with Applications 3 (1996) 413–426.

[29] Netlib, A collection of mathematical software, papers, and databases at http://www.netlib.org.

[30] B. Plateau, On the stochastic structure of parallelism and synchronization models for distributed algorithms, in: Proceedings of the ACM SIGMETRICS Conference on Measurement and Modelling of Computer Systems, Austin, Texas, 1985, pp. 147–154.

[31] B. Plateau, K. Atif, Stochastic automata network for modeling parallel systems, IEEE Transactions on Software Engineering 17 (1991) 1093–1108.

[32] B. Plateau, J.-M. Fourneau, A methodology for solving Markov models of parallel systems, Journal of Parallel and Distributed Computing 12 (1991) 370–387.

[33] G.W. Stewart, Matrix Algorithms, Vol II: Eigensystems, SIAM Press, Philadelphia, Pennslyvania, 2002.

[34] W.J. Stewart, Introduction to the Numerical Solution of Markov Chains, Princeton University Press, Princeton, New Jersey, 1994.

[35] E. Uysal, T. Dayar, Iterative methods based on splittings for stochastic automata networks, European Journal of Operational Research 110 (1998) 166–186.

[36] H.A. van der Vorst, BI-CGSTAB: A fast and smoothly converging variant of BI-CG for the solution of nonsymmetric linear systems, SIAM Journal on Scientific and Statistical Computing 13 (1992) 631–644.

[37] C.F. Van Loan, The ubiquitous Kronecker product, Journal of Computational and Applied Mathematics 123 (2000) 85–100.

[38] C.M. Woodside, Y. Li, Performance Petri net analysis of communications protocol software by delay equivalent aggregation, in: Proceedings of the 4th International Workshop on Petri Nets and Performance Models, IEEE CS-Press, 1991, pp. 64–73.