Computing 73, 349–371 (2004)
Digital Object Identifier (DOI) 10.1007/s00607-004-0074-2

Computing
Printed in Austria

Comparison of Multilevel Methods for Kronecker-based Markovian Representations

P. Buchholz, Dortmund and **T. Dayar**, Ankara

Received June 24, 2003; revised April 8, 2004
Published online: July 26, 2004

Abstract

The paper presents a class of numerical methods to compute the stationary distribution of Markov chains (MCs) with large and structured state spaces. A popular way of dealing with large state spaces in Markovian modeling and analysis is to employ Kronecker-based representations for the generator matrix and to exploit this matrix structure in numerical analysis methods. This paper presents various multilevel (ML) methods for a broad class of MCs with a hierarchcial Kronecker structure of the generator matrix. The particular ML methods are inspired by multigrid and aggregation-disaggregation techniques, and differ among each other by the type of multigrid cycle, the type of smoother, and the order of component aggregation they use. Numerical experiments demonstrate that so far ML methods with successive over-relaxation as smoother provide the most effective solvers for considerably large Markov chains modeled as HMMs with multiple macrostates.

AMS Subject Classifications: 65F10 (primary), 60J27 (secondary).

Keywords: Multilevel methods, multigrid, aggregation-disaggregation, Markov chains, Kronecker-based numerical techniques.

1. Introduction

Markov chains (MCs) are a commonly used mathematical model to describe the quantitative behavior of discrete event systems. Usually some high-level formalisms like Stochastic Petri Nets (SPNs) or Queueing Networks (QNs) are used to specify a model which is afterwards mapped onto a MC. Often the stationary distribution of the MC is computed with numerical methods to determine performance or dependability measures of the modeled system [25]. Although the computation of the stationary distribution requires nothing more than the solution of a set of linear equations, practical problems arise due to the enormous size of the state space of MCs resulting from realistic examples which often grows exponentially with the model specification. A popular way of dealing with this so called "state space explosion problem" is to employ a Kronecker [28] (or tensor) based representation of the generator matrix of the MC which remains compact even for considerably large state spaces.

In the Kronecker-based approach, the system of interest is modeled so that it is formed of smaller interacting components, and its larger underlying MC is neither

generated nor stored but rather represented using Kronecker products of the smaller component matrices. This introduces considerable storage savings at the expense of some overhead in the analysis phase. The concept of using Kronecker operations to define large MCs underlying structured representations is very natural for many application areas since complex systems are usually composed of interacting components. It appears in specification techniques like hierarchical Markovian models (HMMs) [4], [10], [12], or in compositional Markovian models such as stochastic automata networks (SANs) [22], [23], [15] and different classes of superposed Stochastic Petri Nets (SPNs) [14], [19]. In order to analyze large, structured Markovian models efficiently, various algorithms for vector-Kronecker product multiplication are devised [15], [11] and used as kernels in iterative solution techniques proposed for HMMs [4], [5], [8], [9], SANs [25], [26], [5], [11], and superposed Generalized SPNs [19].

In this paper, we consider the steady-state analysis of HMMs, which consist of multiple low level models (LLMs) and a high-level model (HLM) that defines the interaction among LLMs. The HMM formalism is a natural way to describe complex systems and can be interpreted as an extension of well known specification formalisms like QNs, SPNs, or a subclass of SANs which are enhanced by some information about a hierarchical structure. We introduce the specification formalism here only by means of a simple example; detailed information can be found in the literature [4], [5], [10], [12]. However, it should be mentioned that almost all MCs resulting from practical applications can be represented as HMMs and this representation can be easily derived from the model specification using an appropriate modeling tool [3].

Our aim is to solve

$$\pi Q = 0, \quad \sum_{i=0}^{n-1} \pi_i = 1, \tag{1}$$

where Q is the infinitesimal generator or generator matrix (i.e., continuous-time Markov chain, CTMC) of order n underlying an HMM and π is its (row) stationary probability vector. We number the states of Q starting from 0 and assume it is irreducible implying π is also its steady-state vector [25]. The matrix Q has nonnegative off-diagonal elements known as (exponential) transition rates and diagonal elements that are negated row sums of its off-diagonal elements; hence, Q has row sums of zero. We remark that Q is a nonsymmetric matrix, in many cases having nonzero elements of different magnitudes, and arises in application areas such as communication systems, computer systems, and manufacturing systems. Equation (1) can be viewed as a homogeneous linear system with a singular coefficient matrix of rank $(n-1)$ subject to a normalization condition so that its solution vector can be uniquely determined. From π various result measures of performance or dependability can be derived.

The CTMC underlying an HMM can be expressed using sums of Kronecker products thereby facilitating the representation of considerably large Markovian models compactly. It is very important to note that the (nonzero) elements of Q

underlying the Kronecker representation are never explicitly generated, and iterative solvers geared towards such representations utilize a specific vector-Kronecker product multiplication algorithm [15]. Hence, one has a number of constraints to consider that do not exist when working with flat, sparse matrices. Recall that we work with very large nonsymmetric matrices that are stored very compactly. Typically the number of LLMs in an HMM is at least three, implying a problem dimension of at least four when there are multiple macrostates. Different numerical solvers have been implemented and tested for HMMs [5]. The most effective solvers known for HMM problems with multiple macrostates and of dimension four or larger are block successive over-relaxation (BSOR) [8] preconditioned BiConjugate Gradient STABilized (BiCGStab) [27] and Transpose Free Quasi-Minimal Residual (TFQMR) [16] methods (see [9]) as recently shown empirically by comparing different solvers on a large number of examples. Unfortunately, solvers using BSOR are sensitive to the ordering of LLMs, the block partitionings chosen, and the amount of fill-in in the factorized diagonal blocks so that a robust implementation for arbitrary models is difficult to achieve. This paper aims at improving the state-of-the-art in steady-state solvers for such HMMs using ideas from multigrid [17], [29] and aggregation-disaggregation [25] techniques.

Multigrid (MG) techniques are iterative algorithms defined on multiple grids of increasing coarseness for the problem at hand through which the solution process proceeds in cycles until some predetermined stopping criteria are met. One cycle of MG consists of the traversal of these grids from the finest to the coarsest and back to the finest in some order. The finest grid is where the solution vector is required. The coarser grids are where smaller, approximate versions of the original problem are solved.

The general MG algorithm may be formulated recursively (see the multigrid algorithm in [24]). As boundary case, a linear system at the coarsest grid is solved. At intermediate, finer grids there are a number of consecutive recursive calls to the next coarser grid which are preceded by smoothing, residual computation, and restriction operations and are followed by interpolation (or prolongation), correction, and smoothing operations. We will refer to smoothing operations before the recursive call(s) as pre-smoothing iterations, those after as post-smoothing iterations, and the method used in the process as the smoother. At an intermediate grid, for a V-cycle (which is the standard) the number of recursive calls to the next coarser grid is one, whereas that for a W-cycle is two. An F-cycle at an intermediate grid is slightly more complicated, but can be viewed as a recursive call to a W-cycle followed by a recursive call to a V-cycle on the next coarser grid. The MG idea has been shown to provide effective solvers for (partial) differential equations when the grids are chosen appropriately.

A multilevel (ML) algorithm inspired by MG has been presented for the steady state analysis of large, sparse MCs in [18]. Therein the restriction and interpolation operations of MG are replaced respectively with aggregation and disaggregation [25], [20], and accordingly residual computation and correction operations are

omitted. The number of grids employed in the corresponding ML solver is about $log_2 n$ owing it to fact that the number of unknowns in each grid is halved at the next coarser grid. Unfortunately, this solver is hindered by the sparse generation and storage of the intermediate aggregated matrices for general MC problems, which restricts the size of solvable MCs to relatively small state spaces much smaller than the examples solved with the method presented here. Additionally, the method is often not more efficient than SOR due to the overhead for the computation of the aggregated matrices in each ML cycle (see [7] for an empirical comparison).

The Kronecker structure of an HMM suggests a natural definition for the grids. Since HMM components form a hierarchy, one grid can be associated with each level of the hierarchy implying as many grids as the number of LLMs plus one for the HLM when it has multiple states (that is, the case of multiple macrostates). With this choice of grids, if one replaces the restriction and interpolation operations respectively with aggregation and disaggregation, then residual computation and correction operations disappear from the MG algorithm as in [18] and one can still utilize the Kronecker structure on intermediate aggregated matrices. This has the advantage that generator matrices of CTMCs at intermediate grids never need to be generated since they are defined by slightly modifying the Kronecker representation. Such a view formed the basis of the ML algorithm in [6] for HMMs with one macrostate whose generators can be represented as sums of Kronecker products. The particular ML solver therein employed a V-cycle and could use either the power or the Jacobi over-relaxation (JOR) method as smoother.

This paper extends the ML solver in [6] to HMMs with multiple macrostates and with the capability of using (V, W, F) cycles, (power, JOR, SOR) methods as smoothers, and (fixed, cyclic, dynamic) orders in which LLMs can be aggregated in a cycle. Then it provides the results of numerical experiments on a set of HMMs showing that the ML method with SOR smoother provides the most effective solver for HMMs with multiple macrostates so far and is in fact the most effective solver currently available for considerably large CTMCs with a generator that cannot be held in sparse format in main memory. Even for smaller CTMCs where the generator can be stored as a sparse matrix in main memory, ML solvers exploiting the Kronecker representation demonstrate a performance comparable to the performance of the most effective solvers for sparse matrices [7]. The storage requirements of the proposed solver can be forecasted from the HMM description and are nearly insensitive to the ordering of LLMs to the contrary of BSOR preconditioned projection methods. Note that in the proposed solver, the smoother is used with matrices that are recomputed at each cycle and held in Kronecker form. The nonzero elements in these matrices are never explicitly generated. Therefore, we are restricted to using smoothers that are formulated for sums of Kronecker products, that are not based on factorizations, and that do not benefit from the values of the nonzero elements in the matrices. In other words, the smoother should be as simple as possible.

The next section introduces the Kronecker-based description of CTMCs underlying HMMs on a model from the literature. The third section presents the proposed class of ML methods for HMMs with multiple macrostates and discusses how they work. The fourth section provides results of numerical experiments. The fifth section concludes the paper.

2. Hierarchical Markovian Models

We introduce HMMs on an example from the literature so that the inherent structure of the Kronecker representation and ideas related to multigrid and iterative aggregation-disaggregation can be closely observed. Yet, the interested reader can find a formal definition and detailed examples of HMMs in [5, pp. 387–390]. We recall that the Kronecker product of two matrices $A \in \mathbb{R}^{r_A \times c_A}$ and $B \in \mathbb{R}^{r_B \times c_B}$ results in the matrix $U \in \mathbb{R}^{r_A r_B \times c_A c_B}$, which is written as $U = A \otimes B$ and whose elements satisfy $u_{i_A r_B + i_B, j_A c_B + j_B} = a_{i_A, j_A} b_{i_B, j_B}$ [28]. The Kronecker sum of two square matrices $E \in \mathbb{R}^{r_E \times r_E}$ and $F \in \mathbb{R}^{r_F \times r_F}$ results in the matrix $V \in \mathbb{R}^{r_E r_F \times r_E r_F}$, which is written as $V = E \oplus F$ and defined in terms of two Kronecker products as $V = E \otimes I_{r_F} + I_{r_E} \otimes F$. Here I_{r_E} and I_{r_F} respectively denote identity matrices of orders r_E and r_F. Oberserve that Kronecker operations are asssociative such that they can be naturally defined for more than two matrices. Kronecker operations realize a linearization of a multi-dimensional state space. Thus, the Kronecker product/sum of K matrices describes transitions in a K-dimensional state space which are mapped onto transitions in a one-dimensional state space [28]. Observe that the Kronecker product $A \otimes B$ is an $r_A \times c_A$ block matrix. Therefore, it can be perceived as a two-grid. The Kronecker product of K matrices defines a nested block partitioning having $(K-1)$ intermediate levels and can be perceived as a K-grid.

Example 1: We consider a model of the multiserver multiqueue discussed in [1] and name it as *msmq_medium*. The model consists of the HLM and five LLMs, each corresponding to a finite queue of capacity 5. Customers arrive at each queue according to a Poisson process and those that arrive at a full queue get lost. There are 2 servers serving the 5 queues in a round-robin manner. The state space of the HLM is defined by considering the distribution of servers among the LLMs, resulting in 15 HLM states (two servers distributed among 5 LLMs). When a server arrives at a queue with customers, it serves the customer at the head of the queue and travels to the next queue in line. A server that arrives to an empty queue, immediately moves to the next queue in line. Service times at queues and traveling times from one queue to the next are exponentially distributed. Note that there can be two servers simultaneously serving two different customers at the same queue. The structure of the model is shown in Fig. 1. The figure depicts the state in which the first server is serving the first queue and the other server travels from the fourth to the fifth queue. Each LLM describes a queue together with the arrival and service process, and has 32 states. The state space of each LLM is partitioned into three subsets depending on the number of

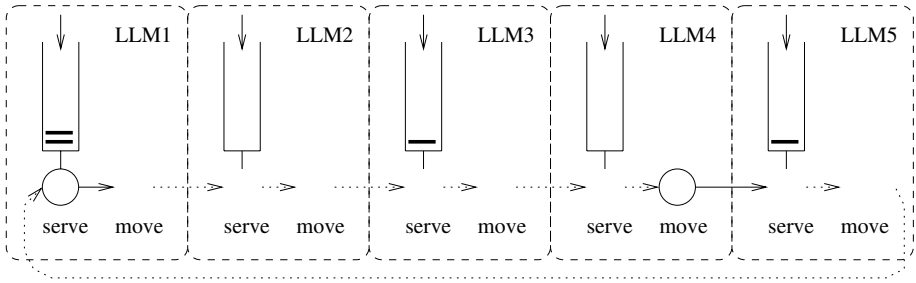

Fig. 1. Structure of the MSMQ example

servers momentarily serving the corresponding queue or traveling from the particular queue to the next in line. With zero servers at the queue we have 6 LLM states describing the queue with population 0 through 5, with one server we have 11 states (i.e., 6 states when the server travels to the next queue and 5 states when the server is serving which requires at least one customer), and with two servers we have 15 states. All states are numbered 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. Macrostates in an HLM may have different numbers of microstates when LLMs have partitioned state spaces, as in this example. For more information about the HMM components in this example and the corresponding matrices, see [5, pp. 390–392].

Six transitions denoted by t_0, t_1, t_2, t_3, t_4, and t_5 take place in the HLM and affect the LLMs. Transition t_0 covers all local transitions inside the LLMs, whereas transitions t_i describe the movement of a server from queue i to $i+1$, for $0 < i < 5$, and from queue 5 to 1, when $i = 5$. The transitions $t_1, \ldots, t_5$ are captured by the following (15×15) HLM matrix which will define the coarsest grid in the ML solver:

Table 1. Mapping between LLM states and HLM states in *msmq_medium*

1HLM	LLM 1	LLM 2	LLM 3	LLM 4	LLM 5	# of microstates
0	17:31	0:5	0:5	0:5	0:5	15. 6. 6. 6. 6 = 19,440
1	6:16	6:16	0:5	0:5	0:5	11. 11. 6. 6. 6 = 26,136
2	6:16	0:5	6:16	0:5	0:5	11. 6. 11. 6. 6 = 26,136
3	6:16	0:5	0:5	6:16	0:5	11. 6. 6. 11. 6 = 26,136
4	6:16	0:5	0:5	0:5	6:16	11. 6. 6. 6. 11 = 26,136
5	0:5	17:31	0:5	0:5	0:5	6. 15. 6. 6. 6 = 19,440
6	0:5	6:16	6:16	0:5	0:5	6. 11. 11. 6. 6 = 26,136
7	0:5	6:16	0:5	6:16	0:5	6. 11. 6. 11. 6 = 26,136
8	0:5	6:16	0:5	0:5	6:16	6. 11. 6. 6. 11 = 26,136
9	0:5	0:5	17:31	0:5	0:5	6. 6. 15. 6. 6 = 19,440
10	0:5	0:5	6:16	6:16	0:5	6. 6. 11. 11. 6 = 26,136
11	0:5	0:5	6:16	0:5	6:16	6. 6. 11. 6. 11 = 26,136
12	0:5	0:5	0:5	17:31	0:5	6. 6. 6. 15. 6 = 19,440
13	0:5	0:5	0:5	6:16	6:16	6. 6. 6. 11. 11 = 26,136
14	0:5	0:5	0:5	0:5	17:31	6. 6. 6. 6. 15 = 19,440

$$
\begin{array}{c}
\\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \\ 11 \\ 12 \\ 13 \\ 14
\end{array}
\begin{array}{c}
\begin{array}{ccccccccccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 \end{array} \\
\begin{pmatrix}
 & t_1 & & & & & & & & & & & & & \\
 & & t_2 & & & t_1 & & & & & & & & & \\
 & & & t_3 & & & t_1 & & & & & & & & \\
 & & & & t_4 & & & t_1 & & & & & & & \\
t_5 & & & & & & & & t_1 & & & & & & \\
 & & & & & & t_2 & & & & & & & & \\
 & & & & & & & t_3 & & t_2 & & & & & \\
 & & & & & & & & t_4 & & t_2 & & & & \\
t_5 & & & & & & & & & & & t_2 & & & \\
 & & & & & & & & & & t_3 & & & & \\
 & & & & & & & & & & & t_4 & t_3 & & \\
 & & t_5 & & & & & & & & & & & t_3 & \\
 & & & & & & & & & & & & & t_4 & \\
 & & & t_5 & & & & & & & & & & & t_4 \\
 & & & & t_5 & & & & & & & & & &
\end{pmatrix}
\end{array}
. \qquad (2)
$$

To each transition in the HLM matrix corresponds a Kronecker product of five (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 and t_5 respectively with the matrices $Q_{t_1}^{(1)}$ and $Q_{t_5}^{(1)}$; LLM 2 participates in t_1 and t_2 respectively with the matrices $Q_{t_1}^{(2)}$ and $Q_{t_2}^{(2)}$; LLM 3 participates in t_2 and t_3 respectively with the matrices $Q_{t_2}^{(3)}$ and $Q_{t_3}^{(3)}$; LLM 4 participates in t_3 and t_4 respectively with the matrices $Q_{t_3}^{(4)}$ and $Q_{t_4}^{(4)}$; and LLM 5 participates in t_4 and t_5 respectively with the matrices $Q_{t_4}^{(5)}$ and $Q_{t_5}^{(5)}$. In general, these matrices are very sparse and therefore held in row sparse format [25]. In this example, each of the transitions t_1, t_2, t_3, t_4, t_5 affects exactly two LLMs. For instance, the Kronecker product associated with t_5 in element (4,0) of the HLM matrix in Eq. (2) is

$$Q_{t_5}^{(1)}(6:16, 17:31) \otimes I_6 \otimes I_6 \otimes I_6 \otimes Q_{t_5}^{(5)}(6:16, 0:5),$$

where $Q_{t_5}^{(1)}(6:16, 17:31)$ denotes the submatrix of $Q_{t_5}^{(1)}$ that lies between states 6 through 16 rowwise and states 17 through 31 columnwise, I_6 denotes the identity matrix of order 6, $Q_{t_5}^{(5)}(6:16, 0:5)$ denotes the submatrix of $Q_{t_5}^{(5)}$ that lies between states 6 through 16 rowwise and states 0 through 5 columnwise. Hence, this particular Kronecker product yields a $(26,136 \times 19,440)$ matrix. The rates associated with the 25 transitions in (2) are all 1 in this example. The transition rates are scalars that multiply the corresponding Kronecker products.

Other than Kronecker products due to the transitions in (2), there is a Kronecker sum implicitly associated with each diagonal element of the HLM matrix. Each Kronecker sum is formed of five LLM matrices corresponding to *local transition* t_0. For instance, the Kronecker sum associated with element (5,5) of the HLM matrix is

$$Q_{t_0}^{(1)}(0:5,0:5) \oplus Q_{t_0}^{(2)}(17:31,17:31) \oplus Q_{t_0}^{(3)}(0:5,0:5) \oplus Q_{t_0}^{(4)}(0:5,0:5) \oplus Q_{t_0}^{(5)}(0:5,0:5).$$

Each Kronecker sum is a sum of five Kronecker products in which all but one of the matrices are identity [25]. 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 is the reason behind naming t_0 a local transition. The particular Kronecker sum associated with element (5,5) of the HLM matrix is $(19,440 \times 19,440)$.

In the HLM matrix of *msmq_medium*, there do not exist any non-local transitions along the diagonal. In general, this need not be so. Therefore, we introduce the following definition.

Definition 1: *In a given HMM, let K be the number of LLMs,* $\mathscr{S}_j^{(k)}$ *be the subset of states of LLM k mapped to macrostate j,* $\mathscr{T}_{i,j}$ *be the set of LLM non-local transitions in element* (i,j) *of the HLM matrix,* $rate_{t_e}(i,j)$ *be the rate associated with transition* $t_e \in \mathscr{T}_{i,j}$*, and* D_j *be the diagonal (correction) matrix that sums the rows of Q corresponding to macrostate j to zero. Then the diagonal block* (j,j) *of Q corresponding to element* (j,j) *of the HLM matrix is given by*

$$Q_{j,j} = \bigoplus_{k=1}^{K} Q_{t_0}^{(k)}(\mathscr{S}_j^{(k)}, \mathscr{S}_j^{(k)}) + \sum_{t_e \in \mathscr{T}_{j,j}} rate_{t_e}(j,j) \bigotimes_{k=1}^{K} Q_{t_e}^{(k)}(\mathscr{S}_j^{(k)}, \mathscr{S}_j^{(k)}) + D_j, \qquad (3)$$

and, when there are multiple macrostates, the off-diagonal block (i,j) *of Q corresponding to element* (i,j) *of the HLM matrix is given by*

$$Q_{i,j} = \sum_{t_e \in \mathscr{T}_{i,j}} rate_{t_e}(i,j) \bigotimes_{k=1}^{K} Q_{t_e}^{(k)}(\mathscr{S}_i^{(k)}, \mathscr{S}_j^{(k)}). \qquad (4)$$

When there are multiple macrostates, Q is a block matrix having as many blocks in each dimension as the number of macrostates (i.e., order of the HLM matrix). The diagonal of Q is formed of its negated off-diagonal row sums, and may be stored explicitly or can be generated as needed.

Each row/column in matrix Q has a two-dimensional address, namely the block number and the state number in the block. At the model level, state (i,x) with macrostate i and detailed state x is characterized by a $K+$ one-dimensional vector $(i,x_1,\ldots x_K)$ such that $x = \sum_{k=1}^{K} x_k \cdot \left(\prod_{j=k+1}^{K} n_j(i)\right)$, where $n_j(i) = |\mathscr{S}_i^{(j)}|$ and x_k is the number of the state in $\mathscr{S}_i^{(k)}$ when states are number consecutively 0 through $|\mathscr{S}_i^{(j)}| - 1$.

In Example 1, the second term in Eq. (3) is missing. Although Q in *msmq_medium* is of order 358,560 and has 2,135,160 nonzeros, its Kronecker representation needs to store 1 HLM matrix having 25 nonzeros and 15 LLM matrices (since identity matrices are not stored) having a total of 370 nonzeros. This is a substantial saving in storage.

If we neglect the diagonal of Q which is handled separately, from Definition 1 it follows that each nonzero element of the HLM matrix is essentially a sum of Kronecker products. This has a very nice implication on the choice of grids in the proposed ML solver when component aggregation is used in forming the coarser grids. The HLM and LLMs 1 through K define the least coarsest (in other words, the finest) grid. This grid is Q. Since the HLM holds the LLMs together, it will define the coarsest grid, which is not aggregated. Regarding the intermediate grids, let us assume that LLMs are aggregated starting from 1 up to K. Then the HLM and LLMs 2 through K define the first coarser grid when LLM 1 is aggregated. The HLM and LLMs 3 through K define the second coarser grid when LLMs 1 and 2 are aggregated; and so on. We consider a form of aggregation in which all grids are irreducible and have row sums of zero. In the $K+1$-dimensional structure of the CTMC aggregation means to freeze the probability distribution in one dimension and consider in the other dimensions aggregated probability flows according to the frozen dimension. For our example this implies that for an aggregated LLM the distribution of the buffer population is fixed. However, there could be different distributions for each state of the rest of the system.

The Kronecker representation having naturally defined $K+1$ (K) grids when there are multiple (single) macrostate(s) in the HMM, let us concentrate on the sizes of the grids defined by the HLM and LLMs for the assumed order in which LLMs are aggregated. In Example 1, the grids defined in this way by HLM, HLM and LLM 5, HLM and LLMs 4-5, HLM and LLMs 3-5, HLM and LLMs 2-5, HLM and LLMs 1-5 have respectively the sizes (15×15), (119×119), (913×913), $(6,822 \times 6,822)$, $(49,896 \times 49,896)$, $(358,560 \times 358,560)$ (see Table 1 and Eqs. (3)–(4)). Obviously, one is not restricted to aggregating LLMs in the order 1 through K, and can consider other orders. The number of possible orders equals $K!$. In the next section, we introduce the ML method with the grid choices suggested by the Kronecker structure of HMMs and remark that the grids are never explicitly generated.

3. ML Methods for HMMs with Multiple Macrostates

The class of ML methods introduced in this section for HMMs with multiple macrostates have the capability of using (V, W, F) cycles, (power, JOR, SOR) methods as smoothers, and (fixed, cyclic, dynamic) orders in which LLMs can be aggregated in a cycle. These parameters are respectively denoted by C, S, and O. We remark that $C \in \{V, W, F\}$, $S \in \{POWER, JOR, SOR\}$, and $O \in \{FIXED, CYCLIC, DYNAMIC\}$. In a particular ML solver, C, S, and O are fixed at the beginning.

In Algorithm 1, we give the recursive ML function that is invoked for LLMs. It is the driver in Algorithm 2 where the particular ML solver starts executing at the finest grid involving the HLM and all the LLMs, and then invokes the recursive ML function with the order of aggregation in the list $\mathscr{C}$. Each pass through the body of the repeat-until loop in Algorithm 2 corresponds to one cycle of the ML

method. One will notice that steps 3–9 in Algorithm 1 are almost identical to the statements between step 3 and 4 in Algorithm 2. Nevertheless, Algorithm 2 is coded separately since the finest grid is treated somewhat differently as we next explain.

The order of aggregating LLMs in each ML cycle is determined by the list $\mathscr{J}$ defined in Algorithm 2. The elements of $\mathscr{J}$ from its head to its tail are denoted respectively by $\mathscr{J}_1, \mathscr{J}_2, \ldots, \mathscr{J}_{K+1}$. The subscripts of these elements indicate their orders in $\mathscr{J}$. In each ML cycle, HLM is always the last model to be handled due to its special position in the hierarchy. Hence, $\mathscr{J}_{K+1}$ is given the value $K+1$ and is associated with the HLM; this never changes. Initially, LLM k is associated with element $\mathscr{J}_k$ which has the value k for $k = 1, 2, \ldots, K$ (see step 1 of Algorithm 2). In each ML cycle, LLMs are aggregated according to these values starting from the element at the head of the list (see the second statement in the repeat-until loop of Algorithm 2). Hence, LLM $\mathscr{J}_1$ is the first LLM to be aggregated.

In the *FIXED* order of aggregating LLMs, the initial assignment of values to the elements of $\mathscr{J}$ does not change after the ML method starts executing; this is the default order. In the *CYCLIC* order, at the end of each ML cycle a circular shift of elements $\mathscr{J}_1$ through $\mathscr{J}_K$ in the list are performed; this ensures some kind of fairness in aggregating LLMs in the next ML cycle. On the other hand, the *DYNAMIC* order sorts the elements $\mathscr{J}_1$ through $\mathscr{J}_K$ according to the residual norms projected (or restricted) to the corresponding LLM at the end of the ML cycle, and aggregates the LLMs in this sorted order in the next ML cycle (see step 8 of Algorithm 2). This ensures that LLMs which have smaller residual norms are aggregated earlier at finer grids. We expect small residual norms to be indicative of good approximations in those components. Note that at each intermediate grid, the recursive ML function is invoked for the next coarser grid with the list of LLMs in $\mathscr{C}$ which is formed by removing the LLM at the head of the incoming list $\mathscr{D}$ by aggregation (see step 4 in Algorithm 1). Once the list of LLMs is exhausted, that is $K+1$ is the only value remaining in the list $\mathscr{D}$, backtracking from the recursion starts by solving a linear system as large as the HLM matrix (see the first if statement in Algorithm 1).

Algorithm 1: Recursive ML function on LLMs in $\mathscr{D}$

function $ML(Q_{\mathscr{D}}, x_{\mathscr{D}}, \mathscr{D}, \gamma)$
if$(|\mathscr{D}| == 1)$then
  $y_{\mathscr{D}} = \text{solve}(Q_{\mathscr{D}}, x_{\mathscr{D}})$; (step 1)
  if$(C == F)$then (step 2)
    $\gamma = 1$;
else
  $x_{\mathscr{D}} = S(Q_{\mathscr{D}}, x_{\mathscr{D}}, w, MIN\_IN\_PRE, MAX\_IN\_PRE, \rho, RES\_COUNT, \eta_1)$; (step 3)
  $\mathscr{C} = \mathscr{D} - [head(\mathscr{D})]$; (step 4)
  compute $Q_{\mathscr{C}}$ from $Q_{\mathscr{D}}$ and $x_{\mathscr{D}}$; (step 5)

$x_{\mathscr{C}}(s_{\mathscr{C}}) = \sum_{s_{\mathscr{D}} \in \mathscr{S}_{\mathscr{D}}, proj(s_{\mathscr{D}}, \mathscr{D}, \mathscr{C}) == s_{\mathscr{C}}} x_{\mathscr{D}}(s_{\mathscr{D}})$ for all $s_{\mathscr{C}} \in \mathscr{S}_{\mathscr{C}}$; (step 6)

if($\gamma == 1$)then (step 7)

$y_{\mathscr{C}} = ML(Q_{\mathscr{C}}, x_{\mathscr{C}}, \mathscr{C}, \gamma)$;

else

$y_{\mathscr{C}} = ML(Q_{\mathscr{C}}, x_{\mathscr{C}}, \mathscr{C}, \gamma)$;

$y_{\mathscr{C}} = ML(Q_{\mathscr{C}}, x_{\mathscr{C}}, \mathscr{C}, \gamma)$;

$y_{\mathscr{D}}(s_{\mathscr{D}}) = x_{\mathscr{D}}(s_{\mathscr{D}}) \frac{y_{\mathscr{C}}(proj(s_{\mathscr{D}}, \mathscr{D}, \mathscr{C}))}{x_{\mathscr{C}}(proj(s_{\mathscr{D}}, \mathscr{D}, \mathscr{C}))}$ for all $s_{\mathscr{D}} \in \mathscr{S}_{\mathscr{D}}$; (step 8)

$y_{\mathscr{D}} = S(Q_{\mathscr{D}}, y_{\mathscr{D}}, w, MIN\_IN\_POST, MAX\_IN\_POST, \rho, RES\_COUNT, \eta_2)$; (step 9)

return $(y_{\mathscr{D}})$;

Now we discuss the operation that computes the next coarser grid $Q_{\mathscr{C}}$ from the grid $Q_{\mathscr{D}}$ using the vector $x_{\mathscr{D}}$ (see step 5 in Algorithm 1) by aggregating the component at the head of list $\mathscr{D}$ (i.e., $head(\mathscr{D})$) and thus forming the list of components in $\mathscr{C}$. This implies that matrix entries in $Q_{\mathscr{C}}$ depend on vector $x_{\mathscr{D}}$ and will be different in every step of the algorithm. Let $\mathscr{S}_{\mathscr{D}}$ and $\mathscr{S}_{\mathscr{C}}$ respectively denote the state spaces of components in $\mathscr{D}$ and $\mathscr{C}$. A positive number of states $s_{\mathscr{D}} \in \mathscr{S}_{\mathscr{D}}$ are projected to each state $s_{\mathscr{C}} \in \mathscr{S}_{\mathscr{C}}$. We represent this in the algorithms as $\exists s_{\mathscr{D}} \in \mathscr{S}_{\mathscr{D}}, proj(s_{\mathscr{D}}, \mathscr{D}, \mathscr{C}) == s_{\mathscr{C}}$ for all $s_{\mathscr{C}} \in \mathscr{S}_{\mathscr{C}}$. We also remark that there are no unreachable states in HMMs and their underlying CTMCs are always irreducible. Hence, this projection is surjective (or onto).

Algorithm 2: ML driver

main()

$\mathscr{J} = [1, 2, \ldots, K+1]$; $x =$ initial approximation; $it = 0$; $cyc = 0$; $stop = FALSE$; (step 1)

if($C == W$ *or* $C == F$) then (step 2)

$\gamma = 2$;

else

$\gamma = 1$;

repeat (step 3)

$x = S(Q, x, w, MIN\_OUT\_PRE, MAX\_OUT\_PRE, \rho, RES\_COUNT, \nu_1)$;

$\mathscr{C} = \mathscr{J} - [head(\mathscr{J})]$;

compute $Q_{\mathscr{C}}$ from Q and x;

$x_{\mathscr{C}}(s_{\mathscr{C}}) = \sum_{s \in \mathscr{S}, proj(s, \mathscr{J}, \mathscr{C}) == s_{\mathscr{C}}} x(s)$ for all $s_{\mathscr{C}} \in \mathscr{S}_{\mathscr{C}}$;

if($\gamma == 1$) then

$y_{\mathscr{C}} = ML(Q_{\mathscr{C}}, x_{\mathscr{C}}, \mathscr{C}, \gamma)$;

else

$y_{\mathscr{C}} = ML(Q_{\mathscr{C}}, x_{\mathscr{C}}, \mathscr{C}, \gamma)$;

$y_{\mathscr{C}} = ML(Q_{\mathscr{C}}, x_{\mathscr{C}}, \mathscr{C}, \gamma)$;

$y(s) = x(s)\dfrac{y_{\mathscr{C}}(proj(s, \mathscr{J}, \mathscr{C}))}{x_{\mathscr{C}}(proj(s, \mathscr{J}, \mathscr{C}))}$ for all $s \in \mathscr{S}$;

$y = S(Q, y, w, MIN\_OUT\_POST, MAX\_OUT\_POST, \rho, RES\_COUNT, v_2)$;

if$(C == F)$ then (step 4)

$\gamma = 2$;

$x = y; it = it + v_1 + v_2; cyc = cyc + 1$; (step 5)

normalize(x); $r = xQ$; (step 6)

if$(it \geq MAX_IT$ *or* $time \geq MAX\_TIME$ *or* $\|r\| \leq STOP\_TOL)$ (step 7)

$stop = TRUE$;

else

if$(O == DYNAMIC)$ then (step 8)

sort LLM indi- ces $\mathscr{J}_1, \mathscr{J}_2, \ldots, \mathscr{J}_K$ into increasing order of $\|r_k\|$,

where r_k is the residual associated with LLM k and is computed from r;

else if $(O == CYCLIC)$ then

circular_ shift$(\mathscr{J}_1, \mathscr{J}_2, \ldots, \mathscr{J}_K)$;

until$(stop)$;

take x as the steady-state vector π of the HMM;

For each state $s_{\mathscr{C}} \in \mathscr{S}_{\mathscr{C}}$, the columns of the grid $Q_{\mathscr{D}}$ corresponding to the states in $\mathscr{S}_{\mathscr{D}}$ that get projected to the same state $s_{\mathscr{C}}$ are summed. This yields a column aggregated grid whose row sums are zero given that $Q_{\mathscr{D}}$ has row sums of zero. For each state $s_{\mathscr{C}} \in \mathscr{S}_{\mathscr{C}}$, the rows of this column aggregated grid corresponding to the states in $\mathscr{S}_{\mathscr{D}}$ that are projected to the same state $s_{\mathscr{C}}$ are multiplied with the corresponding elements of the row vector $x_{\mathscr{D}}$ and summed. This yields the square grid $Q_{\mathscr{C}}$ which has row sums of zero regardless of the norm of $x_{\mathscr{D}}$. We remark that the grid $Q_{\mathscr{C}}$ is irreducible as long as $x_{\mathscr{D}} > 0$ and $Q_{\mathscr{D}}$ is irreducible (see also [6, pp. 346–348]). In practice, $Q_{\mathscr{C}}$ is not explicitly generated; instead it is represented as a sum of Kronecker products of matrices in $\mathscr{C}$ and an additional vector of a length equal to the number of states in $\mathscr{S}_{\mathscr{C}}$ for each non local transition to capture the effect of aggregated LLMs. A formal representation of the aggregated matrix can be found in [6, p. 347] for HMMs with one macrostate. This representation is valid for every submatrix of $Q_{\mathscr{C}}$ in case of multiple macro states. Note that the tail of $\mathscr{C}$ has the value $K + 1$ corresponding to the HLM. These vectors are used so as to facilitate the operations on the coarser grid $Q_{\mathscr{C}}$ in the recursive ML function. Hence, 25 vectors need to be kept for each intermediate grid except the coarsest in Example 1. Since these vectors are generally much shorter than n, they do not bring considerable storage overhead to the ML method.

The aggregation on the columns of $Q_{\mathscr{D}}$ is also performed on the columns of the row vector $x_{\mathscr{D}}$ yielding the vector $x_{\mathscr{C}} > 0$ when $x_{\mathscr{D}} > 0$ (see step 6 in Algorithm 1). In [6, p. 348] it has been shown that $x_{\mathscr{C}}$ is the stationary vector of $Q_{\mathscr{C}}$ if $x_{\mathscr{D}}$ is the stationary vector of $Q_{\mathscr{D}}$. Step 8 in Algorithm 1 corresponds to the opposite of what is done in step 6; that is, it performs disaggregation using $x_{\mathscr{D}}$, $x_{\mathscr{C}}$, and the newly computed vector $y_{\mathscr{C}}$ to obtain the vector $y_{\mathscr{D}}$. Similar aggregation and disaggregation operations are performed in Algorithm 2 at the finest grid Q.

The variable γ in the two algorithms determines the number of recursive calls to the ML function. In step 2 of the initialization statements before the repeat-until loop in Algorithm 2, γ is set to 2 for a W- or an F- cycle and set to 1 for a V-cycle. After this point, there are two places where the value of γ changes, and these happen only for an F-cycle. Hence, for a V-cycle γ remains 1 and for a W-cycle it remains 2, meaning for V- and W-cycles respectively 1 and 2 recursive calls are made to the ML function on the next coarser grid. On the other hand, for an F-cycle γ is set to 1 at the boundary case of the recursion (see step 2 in Algorithm 1). Recall that an F-cycle can be seen as a recursive call to a W-cycle followed by a recursive call to a V-cycle. After the F-cycle is over, γ is reset to 2 in step 4 of Algorithm 2 so as to be ready for a new cycle [29, pp. 174–175].

Each ML cycle starts and ends with some number of iterations using the smoother S. See the two statements right after step 3 and right before step 4 in Algorithm 2, respectively. The same is true for each execution of the recursive ML function at intermediate grids as can be seen in steps 3 and 9 of Algorithm 1. The first two arguments of the call to the smoothers in both algorithms represent respectively the grid to be used in the smoothing process and the vector to be smoothed. The user is given the flexibility to specify different numbers of pre- and post-smoothings in the two algorithms. Hence, we have the nonnegative integer pairs of parameters $(MIN\_OUT\_PRE, MAX\_OUT\_PRE)$, $(MIN\_OUT\_POST, MAX\_OUT\_POST)$ for the finest grid handled by Algorithm 2, and (MIN_IN_PRE, MAX_IN_PRE), $(MIN\_IN\_POST, MAX\_IN\_POST)$ for the coarser intermediate grids handled by Algorithm 1.

For each pair of parameters (MIN_*, MAX_*), S performs MAX_* smoothings when $MIN\_* \geq MAX\_*$. When $MIN\_* < MAX\_*$, S performs an adaptive number of smoothings using the two parameters ρ and *RES_COUNT* as follows. Upon entry to the smoother, the residual norm of the current solution vector is computed and recorded. Then MIN_* smoothings are performed and the residual norm of the solution vector is recomputed. If the ratio of the two residual norms is less than ρ, then S stops executing; otherwise, smoothings continue till MAX_* iterations or the ratio of residual norms of two solution vectors *RES_COUNT* iterations apart are less than ρ. Note that the computation of the residual vector requires an extra implicit vector-grid multiply when S is *SOR*. However, this is performed only every *RES_COUNT* smoothings once the smoother is beyond MIN_* smoothings. The parameter w in the call to the smoother is the relaxation parameter for JOR and SOR. The parameters (ν_1, ν_2) and (η_1, η_2) can be used to keep track respectively of the number of (pre-, post-) smoothings at the finest and coarser grids.

We start the ML iteration with x set to the uniform distribution. At the end of each ML cycle the solution vector x is normalized and the residual vector $r = xQ$ is computed. Global convergence of iterative aggregation-disaggregation on CTMCs with the possibility of using iterative methods to solve the aggregated matrices appears in [21]. This result suggests that as long as a sufficient number of smoothings are performed at each grid, the ML method should converge. See also the related comments in [6, p. 350]. The ML iteration continues until the total number of smoothings at the finest grid (i.e., *it*) exceeds *MAX_IT*, the CPU time exceeds *MAX_TIME*, or the residual norm of the solution vector (i.e., $\|r\|$) meets the prespecified stopping tolerance, *STOP_TOL*. At that point, x is taken as the approximation of the steady state vector π. The variable *cyc* counts the number of ML cycles performed until stopping in case this information cannot be obtained from *it*. Note that this is possible for an adaptive number of pre- or post-smoothings at the finest grid. Finally, we remark that the smoothers of choice require two vectors of length n and two vectors (three in *SOR*) as long as the maximum number of microstates per macrostate in the HMM. One of the vectors of length n in *SOR* is required for the computation of residuals in the implementation of *DYNAMIC* ordering of LLMs for aggregation. The next section presents numerical results with the proposed class of ML solvers.

4. Numerical Experiments

We implemented the ML method as discussed in the previous section in C as part of the APNN toolbox [3], [2]. Now the toolbox has three ML solvers named ML_POWER, ML_JOR, and ML_SOR depending on the smoother used. We report the results of three sets of numerical experiments although other experiments that provide results along the same direction have also been performed.

First are the results of the *msmq_medium* problem introduced in Example 1 and the *courier_medium* problem [8], which are discussed in the next subsection. In these medium sized problems, we compare the results of ML solvers among each other and with those of STR_POWER, STR_JOR, STR_RSOR, which respectively implement power, JOR, SOR methods, and also with those of STR_BSOR, BSOR_BICGSTAB, and BSOR_TFQMR available in the APNN toolbox. Here, STR_BSOR is the two-level version of the BSOR solver in [8] that takes advantage of various techniques to reduce the amount of fill-in when factorizing diagonal blocks of the chosen partitioning. The solvers BSOR_BICGSTAB and BSOR_TFQMR are respectively BSOR preconditioned versions of the projection methods BiCGStab and TFQMR in [9]. To the best of our knowledge, BSOR_BICGSTAB and BSOR_TFQMR are the most competitive solvers for HMMs with multiple macrostates when they utilize favorable block partitionings.

Second are the results of experiments that shed light to the scalability of the ML method. These appear in Subsect. 4.2. Having observed in a multitude of experiments that SOR performs best among the three smoothers, we run ML_SOR on different dimensioned versions of the *msmq* problem and compare the results with those of STR_RSOR.

Third are the results of experiments in Subsect. 4.3 on three problems named *msmq_large*, *courier_large*, and *qh_realcontrol*, again with multiple macrostates, using ML_SOR, STR_RSOR, STR_BSOR, BSOR_BICGSTAB, and BSOR_TFQMR. The first two of these problems are larger versions of the two problems used in the first set of experiments in Subsect. 4.1. The medium version of the *msmq* problem has been discussed in detail in Sect. 2. We introduce the medium version of the *courier* problem in Subsect. 4.1.

The characteristics of the three problems used in Subsect. 4.3 are given in Table 2. For each of them, we provide the macrostates (HLM states), the number of nonzeros in the HLM matrix (nz_{HLM}) and their values (rates), the state space partition of each LLM (LLM states), the number of LLM matrices (LLM matrices), the total number of nonzeros in LLM matrices (nz_{LLMs}), the transitions in the off-diagonal part ($\mathscr{T}(i,j), i \neq j$) and the diagonal part ($\mathscr{T}(j,j)$) of the HLM matrix, the number of states (n) and the number of nonzeros (nz) of the underlying CTMC.

The CTMCs underlying all problems in this paper are irreducible. The techniques proposed in [8] to reduce the amount of fill-in require modest storage for the factors of the diagonal blocks in BSOR and BSOR preconditioned projection methods in these problems. Albeit smaller, *qh_realcontrol* is known to be rather difficult to solve. BSOR preconditioned projection methods perform particularly well on this problem. Hence, the results of the last set of experiments will especially indicate the effectiveness of the class of ML solvers.

In each set of experiments we use $w = 1.0$ wherever required, implying Jacobi, Gauss-Seidel (GS), and block GS (BGS) methods rather than JOR, SOR, and BSOR, respectively. Furthermore, as smoother parameters in the ML solvers we let

Table 2. Benchmark problems

Attribute	*msmq_large*	*courier_large*	*qh_realcontrol*
HLM states	$\{0:34\}$	$\{0:12\}$	$\{0:8\}$
nz_{HLM}	75	65	18
	$(rates \in \{10\})$	$(rates \in \{1, 1449.3, 4821.6, 8771.9\})$	$(rates \in \{10,000\})$
LLM 1 states	$\{0:6, 7:19,$ $20:37, 38:59\}$	$\{0:29\}$	$\{0:76, 77:123,$ $124:170, 171:202\}$
LLM 2 states	$\{0:6, 7:19,$ $20:37, 38:59\}$	$\{0, 1:140, 141, 142:201, 202,$ $203:222, 223, 224:227, 228\}$	$\{0:61, 62:99,$ $100:137, 138:163\}$
LLM 3 states	$\{0:6, 7:19,$ $20:37, 38:59\}$	$\{0:321, 322:326, 327:470,$ $471:474, 475:526, 527:529,$ $530:542, 543:544, 545\}$	$\{0:56, 57:91,$ $92:126, 127:150\}$
LLM 4 states	$\{0:6, 7:19,$ $20:37, 38:59\}$	$\{0:14\}$	None
LLM 5 states	$\{0:6, 7:19,$ $20:37, 38:59\}$	None	None
LLM matrices	15	14	15
nz_{LLMs}	790	2,333	1,486
$\mathscr{T}(i,j),\ i \neq j$	$\{t_{14}, t_{15}, t_{16}, t_{17}, t_{18}\}$	$\{t_0, t_{28}, t_{29}, t_{30}\}$	$\{t_{17}, t_{18}, t_{19}, t_{21}, t_{24}, t_{27}\}$
$\mathscr{T}(j,j)$	None	$\{t_{17}, t_{23}\}$	None
n	2,945,880	1,632,600	399,476
nz	19,894,875	9,732,330	1,871,004

$$
\begin{array}{cc}
 & \{(1,1,1,1,1,1,1,1), \\
(MIN\_IN\_PRE, & (3,3,3,3,1,1,1,1), \\
MAX\_IN\_PRE, & (1,1,1,1,3,3,3,3), \\
MIN\_IN\_POST, & (3,3,3,3,3,3,3,3), \\
MAX\_IN\_POST, \quad \in & (5,5,5,5,1,1,1,1), \\
MIN\_OUT\_PRE, & (1,1,1,1,5,5,5,5), \\
MAX\_OUT\_PRE, & (5,5,5,5,5,5,5,5), \\
MIN\_OUT\_POST, & (10,10,10,10,1,1,1,1), \\
MAX\_OUT\_POST) & (1,1,1,1,10,10,10,10), \\
 & (10,10,10,10,10,10,10,10)\}
\end{array}
$$

in the cases where a fixed number of smoothings are performed. Besides, we set $RES\_COUNT = 2$ and $\rho = 0.9$, and let (MIN_IN_PRE, MAX_IN_PRE, $MIN\_IN\_$, $POST$, MAX_IN_POST, MIN_OUT_PRE, MAX_OUT_PRE, MIN_OUT_POST, MAX_OUT_POST) $\in$ $\{(1,3,1,3,1,1,1,1), (1,5,1,5,1,1,1,1), (1,10,1,10,1,1,1,1)\}$ to experiment with an adaptive number of inner pre- and post-smoothing iterations. For each setting of the smoother parameters, we experiment with (V,W,F) cycles and ($FIXED$, $CYCLIC$, $DYNAMIC$) ordering of LLMs for aggregation. Hence, we perform $(10+3)\times 3\times 3 = 117$ experiments per smoother.

All experiments are performed on a 550 MHz Pentium III processor and a 1 GBytes main memory under Linux. The large main memory is necessary due to the large number of vectors of length n used in BSOR preconditioned projection methods. All times are reported as seconds of CPU time. In Tables 4 through 7, we report the times spent in setup and iterative parts of the solvers respectively under columns Setup and Solve, and indicate the fastest solvers in bold. For each smoother, we list the best three ML solvers out of the 117 considered. The it column indicates the number of (outer) iterations it takes the solvers to stop and the res column indicates the infinity norm of the residual upon stopping. We set $MAX\_IT = 1,000$, $MAX\_TIME = 1,000$ seconds, and $STOP\_TOL = 10^{-8}$ in all solvers. In BSOR_BICGSTAB and BSOR_TFQMR, each pass through the body of the code counts as two iterations rather than one since two vector-matrix products are computed. We choose to normalize the solution vector and compute the residual every 10 iterations in the solvers STR_POWER, STR_JOR, STR_RSOR and STR_BSOR.

For the problems in which convergence is observed due to the stopping tolerance of 10^{-8} but the norm of the residual is found to be larger than 10^{-8}, we continued the iterative process by decreasing the stopping tolerance one order of magnitude at a time until we encountered a residual norm less then 10^{-8}. Such a situation is witnessed among BSOR preconditioned projection methods since we work with unnormalized solution vectors and the underlying CTMCs are not scaled. Recall that the system we solve is singular and a non-scaled coefficient matrix with considerably large entries may result in the residual norm being larger than what the (unnormalized) solution vector actually implies (see [13, p. 1697]) especially when convergence takes place rapidly. In only one of the problems we are not able to reduce the residual norm below 10^{-8} by iterating in this manner, and that

happens to be with the BSOR_TFQMR solver in *qh_realcontrol* in Subsect. 4.3, where the residual norm is computed to be 1.2×10^{-8}.

4.1. Performance of ML Solvers on msmq_medium and courier_medium

Other than *msmq_medium*, in this subsection we consider a typical benchmark from the literature which is introduced in [30]. We name this model as *courier_medium*. Its HLM, which has 10 states, describes the interaction among four LLMs. LLM 1 has 15 states, LLM 2 has 217 states, LLM 3 has 88 states, and LLM 4 has 30 states. The state spaces of LLM 2 and 3 are respectively partitioned as 0:1, 2, 3:5, 6:18, 19:22, 23:74, 75:216 and 0, 1, 2:5, 6, 7:26, 27, 28:87. In the particular HLM under consideration, six transitions denoted by t_0 through t_5 take place and affect the LLMs. LLM 2 and 3 each participates in four transitions while each of the other two LLMs participates in one transition. Contrary to *msmq_medium*, the HLM matrix of *courier_medium* has non-local transitions along its diagonal. The rates associated with all HLM transitions are 1. Although the underlying CTMC has 419,400 states and 2,281,620 nonzeros, the Kronecker representation associated with the HMM needs to store 1 HLM matrix having 47 nonzeros and 14 LLM matrices having a total of 845 nonzeros. Note that medium sized problems have in the order of 100,000 states.

In Tables 3 and 4, we provide the results of numerical experiments with the *msmq_medium* and *courier_medium* problems, respectively. The setup times of the ML solvers are all negligible. The *courier_medium* problem seems to be more difficult to solve than *msmq_medium* due to the longer time it takes to be solved by a particular solver, and benefits relatively more from a larger number of pre- and post-smoothings at intermediate grids in ML solvers. The ML_POWER and

Table 3. ML method on *msmq_medium*

Solver	it	res	Setup	Solve
STR_POWER	1,000	10^{-5}	0	303
ML_POWER(3,3,3,3,1,1,1,1), *CYCLIC*, *W*	52	10^{-9}	0	45
ML_POWER(5,5,5,5,1,1,1,1), *CYCLIC*, *W*	46	10^{-9}	0	48
ML_POWER(5,5,5,5,1,1,1,1), *DYNAMIC*, *W*	44	10^{-9}	0	47
STR_JOR	720	10^{-9}	0	237
ML_JOR(3,3,3,3,1,1,1,1), *CYCLIC*, *W*	26	10^{-9}	0	23
ML_JOR(3,3,3,3,1,1,1,1), *CYCLIC*, *F*	32	10^{-9}	0	28
ML_JOR(5,5,5,5,1,1,1,1), *CYCLIC*, *F*	32	10^{-9}	0	32
STR_RSOR	240	10^{-9}	0	104
ML_SOR(1,1,1,1,1,1,1,1), *CYCLIC*, *W*	24	10^{-9}	0	21
ML_SOR(1,1,1,1,1,1,1,1), *CYCLIC*, *F*	22	10^{-9}	0	**19**
ML_SOR(3,3,3,3,1,1,1,1), *CYCLIC*, *W*	18	10^{-9}	0	21
STR_BSOR	120	10^{-9}	2	62
BSOR_BICGSTAB	47	10^{-9}	2	43
BSOR_TFQMR	46	10^{-10}	2	42

Table 4. ML method on *courier_medium*

Solver	it	res	Setup	Solve
STR_POWER	1,000	10^{-2}	1	841
ML_POWER(5,5,5,5,5,5,5,5), *CYCLIC*, *W*	590	10^{-9}	1	693
ML_POWER(5,5,5,5,5,5,5,5), *CYCLIC*, *F*	610	10^{-9}	1	709
ML_POWER(10,10,10,10,10,10,10,10), *CYCLIC*, *W*	640	10^{-9}	1	701
STR_JOR	1,000	10^{-5}	1	924
ML_JOR(1,1,1,1,1,1,1,1), *CYCLIC*, *F*	116	10^{-9}	1	213
ML_JOR(3,3,3,3,1,1,1,1), *CYCLIC*, *V*	106	10^{-9}	1	199
ML_JOR(5,5,5,5,1,1,1,1), *DYNAMIC*, *V*	108	10^{-9}	1	213
STR_RSOR	360	10^{-9}	1	503
ML_SOR(5,5,5,5,1,1,1,1), *CYCLIC*, *V*	28	10^{-9}	1	82
ML_SOR(5,5,5,5,5,5,5,5), *FIXED*, *V*	50	10^{-9}	1	**78**
ML_SOR(5,5,5,5,5,5,5,5), *FIXED*, *W*	50	10^{-9}	1	84
STR_BSOR	60	10^{-10}	4	154
BSOR_BICGSTAB	37	10^{-9}	4	124
BSOR_TFQMR	40	10^{-9}	4	133

ML_SOR solvers in Table 4 also benefit relatively more than those in Table 3 from a larger number of smoothings at the finest grid.

The winner in both problems happens to be an ML_SOR solver, which is significantly better than BSOR_BICGSTAB and BSOR_TFQMR. The winners in *msmq_medium* and *courier_medium* require, respectively, 11 and 10 ML cycles. The quality of the three smoothers are observed to increase in the order *POWER*, *JOR*, and *SOR*. We also see that adaptive number of smoothings do not yield the best ML solvers. This seems to be due to the extra effort spent in computing the residual norms (at least twice in each call to the smoother) to facilitate adaptiveness although a small number of iterations are required of the smoothers for rapid convergence. At least six ML solvers in each of the two tables use *CYCLIC* order of aggregating LLMs. In *msmq_medium* *W* and *F* cycles are used in the best ML solvers, whereas *V* cycle is associated with the two fastest ML_SOR solvers in *courier_medium*. Interestingly, in the more difficult of the two problems, *FIXED* ordering of LLMs for aggregation works better with the two of the three fastest ML_SOR solvers. Finally, we also observe that increasing the number of pre- and post-smoothings at the finest grid tends to reduce the number of ML cycles to convergence (see the ML_POWER and ML_SOR results in Table 4). A similar statement can also be made for smoothings at intermediate grids. However, a smaller number of ML cycles does not necessarily imply a shorter solution time.

4.2. Scalability of ML_SOR

In this subsection, we investigate the scalability of the ML_SOR solver. We consider five HMMs of different dimensions related to the model introduced in Sect. 2. These HMMs are named *msmq_c*3, *msmq_c*4, *msmq_c*5, *msmq_c*6, *msmq_c*7 and have respectively 3, 4, 5, 6, 7 LMMs. Hence, the largest model among these

HMMs with multiple macrostates has a problem dimension of eight. The number of LLMs indicate the number of queues in the corresponding HMM. There are 2 servers serving the queues in each HMM in a round-robin manner as in *msmq_medium*. In all HMMs, each LLM matrix is of order 20 (due to a finite queueing capacity of 3) and the 20 states are partitioned into the three subsets 0–3, 4–10, and 11–9. Each LLM participates in two non-local transitions as in *msmq_medium*. The HLM matrices corresponding to the five HMMs have respectively 6, 10, 15, 21, 28 macrostates and 9, 16, 25, 36, 49 nonzeros. The values of all nonzeros in the HMM matrices are 10. The number of LLM matrices and their total number of nonzeros are respectively 9, 12, 15, 18, 21 and 132, 176, 220, 264, 308. The number of microstates and number of nonzeros in the underlying MCs of the HMMs are respectively 1,020, 7,008, 42,880, 243,456, 1,311,744 and 3,969, 32,976, 235,680, 1,527,552, 9,241,344.

Table 5 presents the results of STR_RSOR and the best three ML_SOR solvers for the five *msmq* problems. Note that, the ML_SOR solvers across all problems in Table 5 have the same parameters. Each of the solvers performs two smoothings at the finest grid, use either *CYLIC* or *DYNAMIC* order of aggregation, and utilize *W* or *F* cycle. The number of ML cycles to convergence range between 11 and 22, but do not vary significantly for a specific ML_SOR solver. On the other hand, the number of iterations performed by STR_RSOR increases as the problem size increases. The setup times for the ML solvers are all negligible. The solution times of *msmq_c*3 and *msmq_c*4 are too small to say something. However, the ratio of solution times of consecutive problems among *msmq_c*5, *msmq_c*6, and *msmq_c*7 seem to resemble the ratio of the number of

Table 5. Scalability of ML_SOR

Problem	Solver	it	res	Setup	Solve
*msmq_c*3	STR_RSOR	180	10^{-9}	0	0
	ML_SOR(1,1,1,1,1,1,1,1), *CYCLIC*, *W*	44	10^{-9}	0	0
	ML_SOR(1,1,1,1,1,1,1,1), *DYNAMIC*, *W*	44	10^{-9}	0	0
	ML_SOR(3,3,3,3,1,1,1,1), *CYCLIC*, *F*	28	10^{-9}	0	0
*msmq_c*4	STR_RSOR	260	10^{-9}	0	1
	ML_SOR(1,1,1,1,1,1,1,1), *CYCLIC*, *W*	40	10^{-9}	0	1
	ML_SOR(1,1,1,1,1,1,1,1), *DYNAMIC*, *W*	38	10^{-9}	0	1
	ML_SOR(3,3,3,3,1,1,1,1), *CYCLIC*, *F*	26	10^{-9}	0	1
*msmq_c*5	STR_RSOR	290	10^{-9}	0	13
	ML_SOR(1,1,1,1,1,1,1,1), *CYCLIC*, *W*	34	10^{-9}	0	4
	ML_SOR(1,1,1,1,1,1,1,1), *DYNAMIC*, *W*	32	10^{-9}	0	4
	ML_SOR(3,3,3,3,1,1,1,1), *CYCLIC*, *F*	24	10^{-9}	0	4
*msmq_c*6	STR_RSOR	360	10^{-9}	0	109
	ML_SOR(1,1,1,1,1,1,1,1), *CYCLIC*, *W*	30	10^{-9}	0	22
	ML_SOR(1,1,1,1,1,1,1,1), *DYNAMIC*, *W*	30	10^{-9}	0	23
	ML_SOR(3,3,3,3,1,1,1,1), *CYCLIC*, *F*	22	10^{-9}	0	23
*msmq_c*7	STR_RSOR	420	10^{-9}	1	873
	ML_SOR(1,1,1,1,1,1,1,1), *CYCLIC*, *W*	30	10^{-9}	1	148
	ML_SOR(1,1,1,1,1,1,1,1), *DYNAMIC*, *W*	30	10^{-9}	1	153
	ML_SOR(3,3,3,3,1,1,1,1), *CYCLIC*, *F*	22	10^{-9}	1	154

states in the corresponding problems. Hence, ML_SOR is clearly scalable in *msmq*.

4.3. Performance of ML_SOR on Benchmark Problems

In this subsection, we consider three problems. The first two problems named *msmq_large* and *courier_large* are respectively larger versions of *msmq_medium* and *courier_medium*, which are analyzed in Subsect. 4.1, and have in the order of 1,000,000 states. The third problem named *qh_realcontrol* is the model of token based scheduling in a queueing network and has in the order of 100,000 states. See Table 2 for characteristics of these problems. The *qh_realcontrol* and *msmq_large* problems do not have any non-local transitions along the diagonal of their HLM matrices, whereas *courier_large* does. Regarding non-local transitions, each LLM in *qh_realcontrol* and *msmq_large* respectively participates in four and two transitions. In *courier_large*, LLM 2 and 3 each participate in four transitions while each of the other two LLMs participate in one transition. The *qh_realcontrol* problem is especially difficult to solve owing it to the existence of nonzeros in its HLM and LLM matrices that have considerably different orders of magnitude.

Table 6. ML_SOR on benchmark problems

Problem	Solver	it	res	Setup	Solve
msmq_large	STR_POWER	280	10^{-3}	3	1,004
	STR_JOR	260	10^{-6}	3	1,030
	STR_RSOR	190	10^{-9}	3	902
	STR_BSOR	120	10^{-9}	24	751
	BSOR_BICGSTAB	46	10^{-9}	24	487
	BSOR_TFQMR	46	10^{-10}	24	470
	ML_SOR(3,3,3,3,1,1,1,1), *CYCLIC*, *W*	14	10^{-9}	3	152
	ML_SOR(3,3,3,3,1,1,1,1), *CYCLIC*, *F*	14	10^{-9}	3	**150**
	ML_SOR(10,10,10,10,1,1,1,1), *CYCLIC*, *V*	14	10^{-9}	3	171
courier_large	STR_POWER	240	10^{0}	3	1,005
	STR_JOR	220	10^{-5}	3	1,017
	STR_RSOR	130	10^{-9}	3	819
	STR_BSOR	48	10^{-7}	21	1,028
	BSOR_BICGSTAB	42	10^{-9}	21	1,020
	BSOR_TFQMR	42	10^{-9}	21	992
	ML_SOR(1,1,1,1,1,1,1,1), *CYCLIC*, *V*	48	10^{-9}	3	483
	ML_SOR(1,1,1,1,1,1,1,1), *CYCLIC*, *W*	44	10^{-9}	3	480
	ML_SOR(1,1,1,1,1,1,1,1), *CYCLIC*, *F*	44	10^{-9}	3	**474**
qh_realcontrol	STR_POWER	1,000	10^{0}	0	432
	STR_JOR	1,000	10^{-3}	0	475
	STR_RSOR	1,000	10^{-4}	0	553
	STR_BSOR	1,000	10^{-4}	6	550
	BSOR_BICGSTAB	276	10^{-9}	6	273
	BSOR_TFQMR	246	10^{-8}	6	**239**
	ML_SOR(5,5,5,5,1,1,1,1), *CYCLIC*, *W*	292	10^{-9}	0	262
	ML_SOR(5,5,5,5,1,1,1,1), *CYCLIC*, *F*	302	10^{-9}	0	271
	ML_SOR(10,10,10,10,1,1,1,1), *CYCLIC*, *W*	272	10^{-9}	0	274

In *msmq_large* and *courier_large*, ML_SOR with *CYCLIC* order of aggregating LLMs provides clear winners (see Table 6). The best ML solvers in *qh_realcontrol* also employ the *CYCLIC* order. The numbers of pre- and post-smoothings at the finest grid performed by the best ML_SOR solvers in all three problems are each one. The numbers of pre- and post-smoothings at the intermediate grids in the winning ML_SOR solvers are three to five. Regarding the type of cycles, the *F* cycle provides two winners among the ML_SOR solvers. It is also the cycle of choice in the second best ML_SOR solver for *qh_realcontrol*. Nevertheless, the *W* cycle appears the most among the nine ML_SOR solvers in Table 6. Note that it takes about 150 cycles to solve *qh_realcontrol* using ML_SOR. For this more difficult problem, BSOR_TFQMR and BSOR_BICGSTAB provide very strong solvers, which are not easy to beat, but the ML solvers demonstrate similar performance.

5. Conclusion

This paper has proposed a class of ML methods for HMMs with multiple macrostates. The ML solvers are capable of using (V, W, F) cycles, (power, JOR, SOR) methods as smoothers, and (fixed, cyclic, dynamic) orders in which LLMs can be aggregated in each cycle. Extensive numerical experiments on three benchmark HMMs have been performed. Results demonstrate that ML with SOR as the smoother provides the most competitive solver for HMMs with multiple macrostates so far. In almost all cases, three to five pre- and post-smoothings at intermediate grids and one pre- and post-smoothing at the finest grid of the ML_SOR solver are sufficient to obtain the solution in a small number of ML cycles. Among the three different orders of aggregating LLMs, the cyclic order seems to be favored the most. Regarding cyle type, W or F can be recommended. The storage requirements of the proposed ML solvers are modest and nearly insensitive to the ordering of LLMs in the given HMM description. This is to the contrary of the situation in BSOR preconditioned projection methods. However, as it is observed in one of the test problems, there may be HMMs that are difficult to solve for which BSOR preconditioned projection methods also provide effective solvers.

Acknowledgments

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. We thank the anonymous referees for their remarks which led to an improved manuscript.

References

[1] Ajmone-Marsan, M., Donatelli, S., Neri, F.: GSPN models of Markovian multiserver multiqueue systems. Performance Evaluation *11*, 227–240 (1990).
[2] APNN-Toolbox case studies: http://www4.cs.uni-dortmund.de/APNN-TOOLBOX/case_studies/

[3] Bause, F., Buchholz, P., Kemper, P.: A toolbox for functional and quantitative analysis of DEDS. In: Quantitative evaluation of computing and communication systems (Puigjaner, R. Savino, N. N., Serra, B., eds.), pp. 356–359. Lecture Notes in Computer Science 1469. Springer 1998.
[4] Buchholz, P.: A class of hierarchical queueing networks and their analysis. Queueing Systems *15*, 59–80 (1994).
[5] Buchholz, P.: Structured analysis approaches for large Markov chains. Appl. Num. Math. *31*, 375–404 (1999).
[6] Buchholz, P.: Multilevel solutions for structured Markov chains. SIAM J. Matrix Anal. Appl. *22*, 342–357 (2000).
[7] Buchholz, P.: Numerical analysis approaches for large Markov chains – experiments, observations, and some new results. Tutorial at the 2003 Illinois Multiconference on Measurement, Modelling, and Evaluation of Computer-Communication Systems, September 2003 (slides are available under *http://ls4-www.cs.uni-dortmund.de/home/buchholz/pb_public.html*).
[8] Buchholz, P., Dayar, T.: Block SOR for Kronecker structured representations. Linear Algebra Appl. *386*, 83–109 (2004).
[9] Buchholz, P., Dayar, T.: Block SOR preconditioned projection methods for Kronecker structured Markovian representations. SIAU J. Sci. Comp. (forthcoming).
[10] Buchholz, P., Kemper, P.: On generating a hierarchy for GSPN analysis. Performance Evaluation Rev. *26*, 5–14 (1998).
[11] Buchholz, P., Ciardo, G., Donatelli, S., Kemper, P.: Complexity of memory-efficient Kronecker operations with applications to the solution of Markov models. INFORMS J. Comp. *12*, 203–222 (2000).
[12] Campos, J., Donatelli, S., Silva, M.: Structured solution of asynchronously communicating stochastic models. IEEE Trans. on Software Engineering *25*, 147–165 (1999).
[13] Dayar, T., Stewart, W. J.: Comparison of partitioning techniques for two-level iterative solvers on large, sparse Markov chains. SIAM J. Sci. Comp. *21*, 1691–1705 (2000).
[14] Donatelli, S.: Superposed stochastic automata: a class of stochastic Petri nets with parallel solution and distributed state space. Performance Evaluation *18*, 21–26 (1993).
[15] Fernandes, P., Plateau, B., Stewart, W. J.: Efficient descriptor-vector multiplications in stochastic automata networks. J. ACM *45*, 381–414 (1998).
[16] Freund, R. W.: A transpose-free quasi-minimal residual method for non-Hermitian linear systems. SIAM J. Sci. Comp. *14*, 470–482 (1993).
[17] Hackbusch, W.: Multi-grid methods and applications. Berlin: Springer 1985.
[18] Horton, G., Leutenegger, S.: A multi-level solution algorithm for steady state Markov chains. Performance Evaluation Rev. *22*, 191–200 (1994).
[19] Kemper, P.: Numerical analysis of superposed GSPNs. IEEE Trans. on Software Engineering *22*, 615–628 (1996).
[20] Krieger, U. R.: Numerical solution of large finite Markov chains by algebraic multigrid techniques. In: Computations with Markov chains (Stewart, W. J., ed.), pp. 403–424. Kluwer Academic Publishers 1995.
[21] Marek, I., Mayer, P.: Convergence analysis of an iterative aggregation/disaggregation method for computing stationary probability vectors of stochastic matrices. Numer. Linear Algebra Appl. *5*, 253–274 (1998).
[22] Plateau, B.: On the stochastic structure of parallelism and synchronization models for distributed algorithms. In: Proc. ACM SIGMETRICS Conf. on Measurement and Modelling of Computer Systems, pp. 147–154, Austin, TX 1985.
[23] Plateau, B., Fourneau, J.-M.: A methodology for solving Markov models of parallel systems. J. Parallel Distrib. Comp. *12*, 370–387 (1991).
[24] Rüde, U.: The multigrid workbench. http://www.mgnet.org/mgnet/tutorials/xwb.html
[25] Stewart, W. J.: Introduction to the numerical solution of Markov chains. Princeton: Princeton University Press 1994.
[26] Uysal, E., Dayar, T.: Iterative methods based on splittings for stochastic automata networks. Europ. J. Operat. Res. *110*, 166–186 (1998).
[27] van der Vorst, H. A.: BI-CGSTAB: A fast and smoothly converging variant of BI-CG for the solution of nonsymmetric linear systems. SIAM J. Sci. Statist. Comp. *13*, 631–644 (1992).
[28] Van Loan, C. F.: The ubiquitious Kronecker product. J. Comp. Appl. Math. *123*, 85–100 (2000).
[29] Wesseling, P.: An introduction to multigrid methods. Chichester: Wiley 1992. http://www.mgnet.org/mgnet-books-wesseling.html

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

Peter Buchholz
Informatik IV
University of Dortmund
44227 Dortmund
Germany
e-mail: peter.buchholz@udo.edu

Tuğrul Dayar
Department of Computer Engineering
Bilkent University
06800 Bilkent, Ankara
Turkey
e-mail: tugrul@cs.bilkent.edu.tr