ANALYZING MARKOV CHAINS
BASED ON KRONECKER PRODUCTS

TUGRUL DAYAR

Abstract. Kronecker products are used to define the underlying Markov chain (MC) in various
modeling formalisms, including compositional Markovian models, hierarchical Markovian models,
and stochastic process algebras. The motivation behind using a Kronecker structured representation
rather than a flat one is to alleviate the storage requirements associated with the MC. With this
approach, systems that are an order of magnitude larger can be analyzed on the same platform.
The developments in the solution of such MCs are reviewed from an algebraic point of view
and possible areas for further research are indicated with an emphasis on preprocessing using reordering,
grouping, and lumping and numerical analysis using block iterative, multilevel, and preconditioned
projection methods.

Key words. Markov chains, Kronecker products, reordering, grouping, lumping, block iterative
methods, multilevel methods, preconditioned projection methods

AMS subject classifications. 60J27, 65F50, 15A72, 65F10, 65B99

1. Introduction. We consider discrete state space, continuous-time Markovian
processes, namely continuous-time Markov chains (CTMCs). A CTMC having
n states may be represented by an \( (n \times n) \) square matrix \( Q \in \mathbb{R}^{n \times n} \) having nonnegative
off-diagonal elements indicating the rates of the exponentially distributed transition
times between pairs of states and a diagonal formed by the negated row sums of its
off-diagonal elements. Therefore, \( Q \) has a nonpositive diagonal and row sums of zero.
This matrix is also known as the infinitesimal generator of the associated Markovian
process [62].

Let \( \pi_0 \in \mathbb{R}^{1 \times n} \) denote the initial probability distribution row vector of \( Q \), where
\( \pi_0 \geq 0 \), \( \pi_0 e = 1 \), and \( e \) is a column vector of ones. Then the transient
probability distribution row vector at time \( t \geq 0 \) is given by

\[
\pi_t = \pi_0 \exp(Qt) = \pi_0 \sum_{i=0}^{\infty} \frac{(Qt)^i}{i!}.
\]

Whenever the steady-state (or limiting, long-run) probability distribution row vector
\( \pi = \lim_{t \to \infty} \pi_t \) exists, it satisfies

\[
\pi Q = 0, \quad \pi e = 1.
\]

Hence, the steady-state distribution is also a stationary distribution [62].

In the Kronecker based approach, \( Q \) is represented using Kronecker products
[30, 65] of smaller matrices and is never explicitly generated. The implementation of
transient solvers for (1.1) and steady-state solvers for (1.2) can rest on this compact
Kronecker representation, thanks to the existence of an efficient vector-Kronecker
product multiplication algorithm known as the shuffle algorithm [30].

In practice, the representation of \( Q \) based on Kronecker products is obtained
using various modeling formalisms. These include compositional Markovian models
such as stochastic automata networks (SANs) [55, 56, 57, 62] and different classes of

---

\*This work is partly supported by the Alexander von Humboldt Foundation.

\†Department of Computer Engineering, Bilkent University, TR-06800 Bilkent, Ankara, Turkey; Tel: (+90) (312) 290-1981; Fax: (+90) (312) 266-4047 (tugrul@cs.bilkent.edu.tr).
superposed stochastic Petri nets [34, 47], hierarchical Markovian models (HMMs) of queueing networks [6], generalized stochastic Petri nets (GSPNs) [22], or systems of asynchronously communicating stochastic modules [24], and stochastic process algebras such as the performance evaluation process algebra (PEPA) [44]. These modeling formalisms are integrated to various software packages such as the Abstract Petri Net Notation (APNN) toolbox [1, 2], the Performance Evaluation of Parallel Systems (PEPS) software tool [4, 54], the PEPA Workbench [29, 53], and the Stochastic Model checking Analyzer for Reliability and Timing (SMART) [28, 61].

An advantage of HMMs is their ability to represent $Q$ using Kronecker products without introducing unreachable states. Matrix diagrams [27] and representations for specific models as in [43] can also be used to achieve the same effect when state spaces are expressed compositionally. There are other approaches that can be used to deal with unreachable states as discussed in [5, 11, 17]. Throughout our discussion, we make the assumption that the MC at hand does not have unreachable states and is irreducible. Yet, in many practical applications, it is very large and has many nonzeros necessitating its storage in memory using Kronecker products. In order to analyze Markovian models based on Kronecker products efficiently, various algorithms for vector-Kronecker product multiplication based on the shuffle algorithm are devised [5, 17, 35, 36, 57, 58] and used as kernels in iterative solution techniques proposed for different modeling formalisms. The transient distribution in (1.1) can be computed through uniformization using vector-Kronecker product multiplications as in [6]. The steady-state distribution in (1.2) also needs to be computed using vector-Kronecker product multiplications, since direct methods based on complete factorizations, such as Gaussian elimination, normally introduce new nonzeros which cannot be accommodated. The two papers [12, 63] provide good overviews of iterative solution techniques for the analysis of MCs based on Kronecker products. Issues related to reachability analysis, vector-Kronecker product multiplication, hierarchical state space generation in Kronecker based matrix representations for large Markov models are surveyed in [23]. A comparison of the merits of the SAN and GSPN modeling formalisms using the PEPS and SMART software packages can be found in [26].

Although Kronecker representations for CTMCs underlying many models of practical applications have been considered, so far only a handful of discrete-time MCs (DTMCs) based on Kronecker products appeared in the literature. For instance, the one in [59] is a model of synchronization via message passing in a distributed system, the one in [56] is a model of the mutual exclusion algorithm of Lamport, those in [37, 38] are models of buffer admission mechanisms for asynchronous transfer mode (ATM) networks from telecommunications, and the one in [66] is a multiservices resource allocation policy for wireless ATM networks. The model in [42] is a larger, scalable, and extended version of that in [66]. It serves as a good example showing that the underlying MC of a discrete-time model based on Kronecker products can be relatively dense and numerically difficult to analyze. These case studies are based on the SAN modeling formalism, whereas [60] extends the Kronecker representation to stochastic Petri nets with discrete phase-type distributions. Clearly, the area of DTMCs based on Kronecker products can use other case studies and formalisms.

Here, we take an algebraic view and discuss recent results related to the analysis of MCs based on Kronecker products independently from modeling formalisms. In section 2, we provide background material on the Kronecker representation of a CTMC, show that it has a rich structure which is nested and recursive, and introduce a small CTMC expressed as a sum of Kronecker products; this CTMC is used as a
running example throughout the discussion. In section 3, we consider preprocessing of the Kronecker representation so as to expedite numerical analysis. We discuss permuting the nonzero structure of the underlying CTMC symmetrically by reordering, changing the orders of the nested blocks by grouping, and reducing the size of the state space by lumping. Sections 4, 5, and 6 are devoted to the steady-state analysis of CTMCs based on Kronecker products with block iterative methods, multilevel methods, and preconditioned projection methods, respectively. In section 7, we conclude. The results can be extended to DTMCs based on Kronecker products with minor modifications. Areas that need further research are mentioned within the sections as they are discussed. In passing, we remark that parallel implementations exploiting the Kronecker representation are beyond the scope of this paper and form an open area for research.

2. Background. Recall that the Kronecker (or tensor) product [30, 65] of two (rectangular) matrices $A \in \mathbb{R}^{n_A \times m_A}$ and $B \in \mathbb{R}^{n_B \times m_B}$ is written as $A \otimes B$ and yields the (rectangular) matrix $C \in \mathbb{R}^{n_A n_B \times m_A m_B}$ whose elements satisfy

$$c(i_C, j_C) = a(i_A, j_A)b(i_B, j_B) \text{ with } i_C = (i_A-1)n_B + i_B \text{ and } j_C = (j_A-1)m_B + j_B$$

for

$$(i_A, j_A) \in \{1, 2, \ldots, n_A\} \times \{1, 2, \ldots, m_A\},$$

$$(i_B, j_B) \in \{1, 2, \ldots, n_B\} \times \{1, 2, \ldots, m_B\},$$

where $\times$ is the Cartesian product operator. Note that in a 2-dimensional representation, the row indices of $C$ are in $\{1, 2, \ldots, n_A\} \times \{1, 2, \ldots, m_A\}$, whereas its column indices are in $\{1, 2, \ldots, m_A\} \times \{1, 2, \ldots, m_B\}$. Hence, the ordering of rows and columns of $C$ with respect to this 2-dimensional representation is lexicographical, since

$$c(i_C, j_C) = c((i_A, i_B), (j_A, j_B)) = c((i_A-1)n_B + i_B, (j_A-1)m_B + j_B).$$

The Kronecker product is associative and defined for more than two matrices. To explain this further for a MC setting, let us consider the Kronecker product of $H$ square matrices as in

$$X = X^{(1)} \otimes X^{(2)} \otimes \cdots \otimes X^{(H)} = \otimes_{h=1}^{H} X^{(h)},$$

where $X^{(h)} \in \mathbb{R}^{n_h \times n_h}$ and row/column indices of $X^{(h)}$ are in $S^{(h)} = \{1, 2, \ldots, n_h\}$ for $h = 1, 2, \ldots, H$. Therefore, $X \in \mathbb{R}^{n \times n}$ with $n = \prod_{h=1}^{H} n_h$, and the ordered $H$-dimensional tuples $(i_1, i_2, \ldots, i_H) \in \times_{h=1}^{H} S^{(h)}$ and $(j_1, j_2, \ldots, j_H) \in \times_{h=1}^{H} S^{(h)}$ may be used to represent the row and column indices of $X$, respectively. Hence, the Kronecker product of $H$ square matrices implies a one-to-one onto mapping between an $H$-dimensional state space and a one-dimensional state space that are lexicographically ordered, and naturally the Kronecker product has been used to define MCs having multi-dimensional state spaces.

2.1. Kronecker representation of $Q$. Without going into detail, we assume that the $H$-dimensional CTMC at hand is represented as a sum of Kronecker products plus a diagonal matrix. Specifically, we have

$$Q = Q_O + Q_D, \quad Q_O = \sum_{k=1}^{K} \bigotimes_{h=1}^{H} Q^{(h)}_k, \quad Q_D = \text{diag}(-Q_O e),$$

(2.1)
where $Q_O$ and $Q_D$ correspond respectively to the off-diagonal and diagonal parts of $Q$, $K$ is the number of Kronecker products (or terms) forming $Q_O$, $H$ is the number of factors in each Kronecker product, $Q_k^{(h)} \in \mathbb{R}^{n_h \times n_h}$ and satisfies $Q_k^{(h)} \geq 0$ for $k = 1, 2, \ldots, K$ and $h = 1, 2, \ldots, H$, and diag is used to denote a diagonal matrix which has its vector argument along its diagonal. Observe that $Q_O \geq 0$ and $Q_D \leq 0$.

If row/column indices of $Q_k^{(h)}$ are in $S^{(h)} = \{1, \ldots, n_h\}$ for $k = 1, 2, \ldots, K$ and $h = 1, 2, \ldots, H$, then the $H$-dimensional state space of $Q$ is given by $S = \times_{h=1}^H S^{(h)}$.

Observe that $|S| = \prod_{h=1}^H |S^{(h)}| = \prod_{h=1}^H n_h = n$. Furthermore, the one-dimensional value of state $s \in S$ corresponding to $(s_1, s_2, \ldots, s_H)$, where $s_h \in S^{(h)}$ for $h = 1, 2, \ldots, H$, is given by

$$s = 1 + \sum_{h=1}^H (s_h - 1) \prod_{i=h+1}^H n_i.$$ 

Throughout the text, we will be using the one-dimensional and multi-dimensional representations of states interchangeably.

One needs space for the diagonal matrix $Q_D$ and the matrices in the Kronecker representation of $Q_O$, meaning a floating-point vector of length $\prod_{h=1}^H n_h$ and at most $K$ (sparse) floating-point matrices of order $n_h$ are stored for $h = 1, 2, \ldots, H$. In the worst case, this amounts to a storage space of $n + \sum_{h=1}^H nz_{Q_k^{(h)}}$ floating-point values, where $nz_{Q_k^{(h)}}$ is the sum of the number of nonzeros in $Q_k^{(h)}$ for $k = 1, 2, \ldots, K$, compared to $nz$ nonzeros required by the flat representation. We remark that $Q_D$ can also be expressed as a sum of Kronecker products:

$$Q_D = - \sum_{k=1}^K \bigotimes_{h=1}^H \text{diag}(Q_k^{(h)} e).$$

However, in order to enable the efficient implementation of numerical solvers, most of the time $Q_D$ is precomputed and stored explicitly.

The complexity of a vector multiplication with $Q_O$, which consists of $K$ Kronecker product terms, amounts to

$$K \prod_{h=1}^H n_h + 2 \sum_{k=1}^K \prod_{h=1}^H n_h \sum_{l=1}^H nz_{Q_k^{(l)}} / n_l = K \prod_{h=1}^H n_h + 2 \prod_{h=1}^H n_h \sum_{l=1}^H \left( \sum_{k=1}^K nz_{Q_k^{(l)}} \right) / n_l = n(K + 2 \sum_{h=1}^H nz_{Q_k^{(h)}} / n_h)$$

floating-point arithmetic operations [35], where $nz_{Q_k^{(l)}}$ is the number of nonzeros in $Q_k^{(l)}$ for $k = 1, 2, \ldots, K$ and $l = 1, 2, \ldots, H$.

Now, observe that each nonzero element of the matrix $Q_k^{(h)}$ in (2.1) is located by its row and column indices, which are members of $S^{(h)}$. In a more general setting, a nonzero element of $Q_k^{(h)}$ may be a function of states in state spaces other than $S^{(h)}$, thus a function of non-local states. This phenomenon is a by-product of the modeling process and has been utilized in the SAN modeling formalism. These nonzero elements are referred to as functional transitions and the corresponding Kronecker products are said to be generalized [55]. Although it is possible to remove functional transitions from a sum of generalized Kronecker products by introducing new terms [57] and/or
Furthermore, block (\(i\) \(Q\) \(j\)) is nested and recursive [18, 19, 20, 32, 39, 41, 42, 64]. Let level 0 denote the highest level at which \(Q\) is perceived as a single block of order \(n = \prod_{h=1}^{\infty} n_h\). At the next level, which we call level 1, \(Q\) is an \((n_1 \times n_1)\) block matrix with blocks of order \(\prod_{h=2}^{\infty} n_h\). At level 2, \(Q\) is an \((n_1 n_2 \times n_1 n_2)\) block matrix with blocks of order \(\prod_{h=3}^{\infty} n_h\). Continuing in this manner, at level \(H\), \(Q\) is a \((\prod_{h=1}^{H} n_h \times \prod_{h=1}^{H} n_h)\), in other words, \((n \times n)\) block matrix with blocks of order 1. More formally, we have

\[
(2.2) \quad b_l = \begin{cases} 
1, & l = 0 \\
n_l^2 h_{l-1}, & l = 1, 2, \ldots, H 
\end{cases} \quad \text{and} \quad o_l = \begin{cases} 
n, & l = 0 \\
_{o_{l-1}/n_l}, & l = 1, 2, \ldots, H 
\end{cases}
\]

where \(b_l\) and \(o_l\) denote the number and order of blocks at level \(l = 0, 1, \ldots, \).

Unrolling the recurrences, we obtain

\[
b_l = \prod_{h=1}^{l} n_h^2 \quad \text{and} \quad o_l = \prod_{h=l+1}^{H} n_h.
\]

Note that at level \(l\) there are \(\sqrt{b_l}\) blocks each of order \(o_l\) along the diagonal of \(Q\). Furthermore, block \(((i_1, i_2, \ldots, i_l), (j_1, j_2, \ldots, j_l))\) of \(Q\) at level \(l\) is given by

\[
Q((i_1, i_2, \ldots, i_l), (j_1, j_2, \ldots, j_l)) = \sum_{k=1}^{K} \left( \prod_{h=1}^{l} q_k^{(h)}(i_h, j_h) \right) \left( \bigotimes_{h=l+1}^{H} Q_k^{(h)} \right) + Q_D((i_1, i_2, \ldots, i_l), (j_1, j_2, \ldots, j_l))
\]

for \(l = 0, 1, \ldots, H\),

where \(Q_D((i_1, i_2, \ldots, i_l), (j_1, j_2, \ldots, j_l))\) is block \(((i_1, i_2, \ldots, i_l), (j_1, j_2, \ldots, j_l))\) of \(Q_D\) with the understanding that \(l = 0\) yields \(Q\) and \(l = H\) yields the scalar

\[
q((i_1, i_2, \ldots, i_H), (j_1, j_2, \ldots, j_H)) = \sum_{k=1}^{K} \prod_{h=1}^{H} q_k^{(h)}(i_h, j_h) + q_D((i_1, i_2, \ldots, i_H), (j_1, j_2, \ldots, j_H)).
\]

Observe that \(Q_D((i_1, i_2, \ldots, i_l), (j_1, j_2, \ldots, j_l)) = 0\) if \((i_1, i_2, \ldots, i_l) \neq (j_1, j_2, \ldots, j_l)\), meaning it is an off-diagonal block at level \(l\). The nested structure associated with (2.1) is also valid in the presence of functional transitions.

Now, we introduce a small example to illustrate the Kronecker representation of a CTMC.

2.2. An example. Consider the following matrices for a 4-dimensional problem (each dimension with 2 states) having 7 terms of Kronecker products:

\[
Q_1^{(1)} = \left( \begin{array}{cc} 1 & \end{array} \right), \quad Q_2^{(1)} = Q_3^{(1)} = Q_4^{(1)} = Q_5^{(1)} = I, \quad Q_6^{(1)} = \left( \begin{array}{cc} 1 & \end{array} \right), \quad Q_7^{(1)} = \left( \begin{array}{cc} 1 & 1 \\ 0 & \end{array} \right),
\]

\[
Q_1^{(2)} = I, \quad Q_2^{(2)} = \left( \begin{array}{cc} 1 & \end{array} \right), \quad Q_3^{(2)} = Q_4^{(2)} = Q_5^{(2)} = Q_6^{(2)} = I, \quad Q_7^{(2)} = \left( \begin{array}{cc} 1 & \end{array} \right).
\]
\[ Q^{(3)}_1 = Q^{(3)}_2 = I, \quad Q^{(3)}_3 = \begin{pmatrix} 1 \\ 10 \end{pmatrix}, \quad Q^{(3)}_4 = \begin{pmatrix} 1 \\ 10 \end{pmatrix}, \quad Q^{(3)}_5 = I, \quad Q^{(3)}_6 = \begin{pmatrix} 1 \\ 10 \end{pmatrix}, \quad Q^{(3)}_7 = I. \]

Then, from equation (2.1)
\[
Q = \sum_{k=1}^{4} \bigotimes_{h=1}^{4} Q^{(h)}_k + Q_D,
\]
and is given by
\[
Q = \begin{pmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\
1 & 1 & 1 & 1 & 2 & 2 & 2 & 2 & 1 & 1 & 1 & 2 & 2 & 2 & 2 & 2 \\
1 & 2 & 1 & 2 & 1 & 2 & 1 & 2 & 1 & 2 & 1 & 2 & 1 & 2 & 1 & 2 \\
\end{pmatrix}
\]

Now, if we assume the absence of a matrix in the Kronecker representation indicates that it is identity, then it is possible to do without storing identity matrices. With this understanding, the number of floating-point values stored in the Kronecker representation of \( Q \) is 10 for the matrices and 16 for the diagonal elements, thus totaling 26; whereas, it is 60 for the flat representation. The discrepancy between the Kronecker and the flat representations becomes substantial for larger values of the state space size, \( n \). In passing, we remark that it is also possible to take advantage of identity matrices in the vector-Kronecker product multiplication algorithm.

Since \( n = 16 \) and \( n_1 = n_2 = n_3 = n_4 = 2 \), the recursive definition in (2.2) reveals the nested block structure of \( Q \) as follows. At level 0, we have an order 16 matrix; at level 1, we have a \((2 \times 2)\) block matrix with blocks of order 8; at level 2, we have a \((4 \times 4)\) block matrix with blocks of order 4, at level 3, we have an \((8 \times 8)\) block matrix with blocks of order 2; and finally, at level 4, we have a \((16 \times 16)\) block matrix with blocks of order 1.

In the next section, we discuss preprocessing techniques to expedite the analysis of MCs based on Kronecker products.

3. Preprocessing. There are three techniques that can be used to put the Kronecker representation into a more favorable form before solvers take over. These are reordering, grouping, and lumping.

3.1. Reordering and grouping. We assume that \( Q \) is a time-homogeneous CTMC. Therefore, the left-hand side of (2.1) is constant up to a symmetric permutation, that is, up to a reordering of the state space, \( S \). Yet, there may be multiple ways in which the number of Kronecker product terms, \( K \), and the number of factors in each Kronecker product, \( H \), forming \( Q_D \) on the right-hand side of (2.1) are chosen. Obviously, the choice \((K, H) = (1, 1)\) indicates a flat representation and is
assumed to be impossible due to memory limitations. As $H$ decreases towards 1, the
Kronecker representation becomes flatter, implying increased storage requirements.
On the other hand, if $K$ were 1, then $Q$ could be analyzed along each dimension inde-
pendently. Hence, $K$ is normally assumed to be larger than 1. Observe that it would
be advantageous to be able to make $K$ as small as possible without changing $H$, since
then we would be decreasing the number of terms in the Kronecker representation of
$Q_O$ and making the matrices $Q_k^{(h)}$ fuller.

Reordering in MCs based on Kronecker products refers to either permuting the
factors of Kronecker products or renumbering the states in the state spaces of factors.
We remark that the latter corresponds to a symmetric permutation of the factor
matrices $Q_k^{(h)}$ for $k = 1, 2, \ldots, K$ associated with the renumbered state space $S^{(h)}$.
As indicated in [5, 36], reordering of the first kind may be used to reduce the overhead
associated with vector-Kronecker product multiplication in the presence of functional
transitions. Furthermore, reordering of both kinds can change the nonzero structure
of the underlying MC, and thereby can have an effect on the convergence of iterative
methods sensitive to the nonzero structure [31]. Hence, by the help of reordering, it
may be possible to symmetrically permute the nonzero structure of the underlying
MC to a more favorable form for the iterative method of choice. In doing this, we can
use the nonzero structure of $\sum_{k=1}^{K} Q_k^{(h)}$, which indicates how the factor $h$ contributes
to the nonzero structure of $Q_O$ for $h = 1, 2, \ldots, H$.

Grouping in MCs based on Kronecker products refers to collapsing the same adja-
cent factors in each Kronecker product. Consequently, the factors in each Kronecker
product are reduced by the same number and the state space sizes of the factors are
increased. The effect of grouping factors in Kronecker products forming $Q_O$ is investi-
gated in a sequence of papers [5, 35, 36] under functional transitions. The objective is
to reduce the number of factors and thereby the overhead associated with evaluating
functional transitions. Results show that in some cases grouping may help reduce the
state space if it had unreachable states, may decrease the overhead associated with
functional transitions, and may even decrease the number of terms in the Kronecker
representation. When there are functional transitions, the best approach seems to
always group those factors which have functional dependencies among each other. In
the absence of functional transitions, it is recommended to group as many factors as pos-
sible given available memory starting from the highest indexed factor. This ensures a
flatter representation for diagonal blocks at a particular level, which is a useful feature
in certain iterative methods.

The effects of reordering and grouping of factors of Kronecker products on the
convergence and space requirements of iterative methods have been investigated in
a number of papers [18, 20, 32, 39, 64], but a broad, systematic study seems to be
lacking.

In the next section, we recall the concept and types of lumpability, and provide a
brief summary of existing work associated with lumping on MCs based on Kronecker
products.

3.2. Lumpability. Lumpability [46] is a property possessed by some MCs which, if
conditions are met, may be used to reduce a large state space to a smaller one. The
idea is to find a partitioning of the original state space such that, when the states in
each partition are lumped (or aggregated) to form a single state, the resulting MC
described by the lumped states has equivalent behavior to the original chain. It is
therefore important to consider lumping to reduce the size of the state space, $S$, before
moving to the solution phase.
In this work we refer to two kinds of lumpability: ordinary lumpability and exact lumpability. Here we give definitions for CTMCs. Equivalent definitions can be stated for DTMCs. A CTMC $Q$ is said to be \textit{ordinarily lumpable} with respect to a partitioning of its state space $S = \bigcup S_i$ and $S_i \cap S_j = \emptyset$ for all $i \neq j$ if for all $S_i \subset S$ and all $s_i, s_j \in S_i$

![Equation](3.1)

$$\sum_{s_j \in S_j} q(s_i, s_j) = \sum_{s_j \in S_j} q(s_i', s_j) \text{ for all } S_j \subset S.$$  

A CTMC $Q$ is said to be \textit{exactly lumpable} with respect to a partitioning of its state space $S = \bigcup S_i$ and $S_i \cap S_j = \emptyset$ for all $i \neq j$ if for all $S_i \subset S$ and all $s_i, s_j \in S_i$

![Equation](3.2)

$$\sum_{s_j \in S_j} q(s_j, s_i) = \sum_{s_j \in S_j} q(s_j, s_i') \text{ for all } S_j \subset S.$$  

Ordinary lumpability refers to a partitioning of the state space in which the sums of transition rates from each state in a partition to another partition are the same. On the other hand, exact lumpability refers to a partitioning of the state space in which the sums of transition rates from all states in a partition into another partition are the same.

Let $S_{\text{lumped}}$ denote the lumped state space. On the ordinarily lumped MC one can only compute performance measures defined over $S_{\text{lumped}}$. On the exactly lumped MC one can compute steady-state performance measures defined over $S$, transient performance measures defined over $S_{\text{lumped}}$, and transient performance measures defined over $S$ if the states in the exactly lumpable partitions have the same initial probabilities. Since MCs satisfy a row sum property rather than a column sum property, the exact lumpability condition in (3.2) is more difficult to be satisfied than the ordinary lumpability condition in (3.1). See [7] and the references therein for more information regarding the concept of lumpability and its implications.

Lumpability can be investigated on the flat representation of the MC. Detection of ordinary and exact lumpability on $Q$ through partition refinement would imply a time complexity of $O(nz \log n)$ and a space complexity of $O(nz)$ [52]. Since this is expensive in terms of time and storage, techniques that investigate lumpability on the Kronecker representation have been considered.

Lumpability can be investigated within each of the state spaces $S^{(h)}$ that define the Kronecker representation of $Q_O$ in (2.1) for $h = 1, 2, \ldots, H$ independently. For the state space $S^{(h)}$, detection of ordinary lumpability through partition refinement as in [16] requires a time complexity of $O(nz_Q^{(h)} \log n_h)$ and a space complexity of $O(nz_Q^{(h)})$. Then the lumped Kronecker representation may be obtained by replacing each of the state spaces $S^{(h)}$ and its corresponding matrices $Q_k^{(h)}$ for $k = 1, 2, \ldots, K$ with equivalent lumped ones. Lumpability can also be investigated among the state spaces $S^{(h)}$ that are replicated (or identical) with respect to the Kronecker representation of $Q_O$ as in [3]. Therein ordinary lumpability of replicated state spaces is shown in the presence of functional transitions. Note that replication refers to a very specific kind of symmetry in the Kronecker representation, and with ordinary lumpability, only performance measures of interest over $S_{\text{lumped}}$ can be computed. Lumpability can also be investigated among the state spaces $S^{(h)}$ by considering dependencies and matrix properties in the Kronecker representation as in [41, 42]. Therein sufficient conditions that satisfy ordinary lumpability are specified and an iterative steady-state solution method which is able to compute performance
measures over $S$ is given for CTMCs and DTMCs in the presence of functional transitions. The work identifies lumpable partitionings on the underlying MC induced by the nested block structure of the Kronecker representation in (2.2). Although the particular approach of lumping one or more state spaces $S^{(h)}$ totally as in [41, 42] is a very specific kind of performance equivalence and lumping considered in [8, 10], due to its accommodation of functional transitions it also enables the detection of certain ordinarily lumpable partitionings in which blocks are composed of multiple (non-identical) state spaces but the individual state spaces cannot be lumped by themselves. This is not possible with the approaches in [3, 8, 10].

We remark that neither of the two approaches in [3] and [41, 42] that investigate lumping among the state spaces $S^{(h)}$ for $h = 1, 2, \ldots, H$ is completely automated, use a Kronecker representation for the lumped MC, and possess a proper complexity analysis. Furthermore, since the Kronecker representation is rich in structure and the three approaches presented in this section do not work on the flat representation, there can very well be other symmetries in the Kronecker representation which also lead to lumpability. This may be worthwhile investigating.

In the next section, we consider block iterative methods based on splittings for MCs that are in the form of (2.1).

4. Block iterative methods. We begin by splitting the smaller matrices that form the Kronecker products as in [64].

4.1. Splitting the smaller matrices. Let

\begin{equation}
Q_k^{(h)} = D_k^{(h)} + U_k^{(h)} + L_k^{(h)} \quad \text{for} \quad k = 1, 2, \ldots, K \quad \text{and} \quad h = 1, 2, \ldots, H,
\end{equation}

where $D_k^{(h)}$, $U_k^{(h)}$, and $L_k^{(h)}$ are respectively the diagonal, strictly upper-triangular, and strictly lower-triangular parts of $Q_k^{(h)}$. Observe that $D_k^{(h)} \geq 0$, $U_k^{(h)} \geq 0$, and $L_k^{(h)} \geq 0$ since $Q_k^{(h)} \geq 0$. Then using Lemma A.8 in [64, p. 183], which rests on the associativity of Kronecker product and the distributivity of Kronecker product over matrix addition, it is possible to express $Q_O$ of $Q$ in (2.1) at level $l = 0, 1, \ldots, H$ using (4.1) as

\begin{equation}
Q_O = Q_U^{(l)} + Q_L^{(l)} + Q_{DU}^{(l)} + Q_{DL}^{(l)},
\end{equation}

where

\begin{equation}
Q_{U}^{(l)} = \sum_{k=1}^{K} \sum_{h=1}^{l} \left( \bigotimes_{f=1}^{h-1} D_k^{(f)} \right) \otimes U_k^{(h)} \otimes \left( \bigotimes_{f=h+1}^{H} Q_k^{(f)} \right),
\end{equation}

\begin{equation}
Q_{L}^{(l)} = \sum_{k=1}^{K} \sum_{h=1}^{l} \left( \bigotimes_{f=1}^{h-1} D_k^{(f)} \right) \otimes L_k^{(h)} \otimes \left( \bigotimes_{f=h+1}^{H} Q_k^{(f)} \right),
\end{equation}

\begin{equation}
Q_{DU}^{(l)} = \sum_{k=1}^{K} \sum_{h=l+1}^{H} \left( \bigotimes_{f=1}^{h-1} D_k^{(f)} \right) \otimes U_k^{(h)} \otimes \left( \bigotimes_{f=h+1}^{H} Q_k^{(f)} \right),
\end{equation}

correspond respectively to the strictly block upper- and lower-triangular parts of $Q_O$ at level $l$, and
\[ Q_{DL(l)} = \sum_{k=1}^{K} \sum_{h=1}^{H} \left( \bigotimes_{f=1}^{h-1} D_{k}^{(f)} \right) \otimes L_{k}^{(h)} \otimes \left( \bigotimes_{f=h+1}^{H} Q_{k}^{(f)} \right). \]

Correspond respectively to the strictly upper- and lower-triangular parts of the block diagonal of \( QO \) at level \( l \). Observe that \( Q_{U(l)} \geq 0, Q_{L(l)} \geq 0, Q_{DU(l)} \geq 0, \) and \( Q_{DL(l)} \geq 0 \). Furthermore, we remark that \( l = 0 \) implies \( QO \) is a single block for which \( Q_{U(0)} = Q_{L(0)} = 0 \), whereas \( l = H \) corresponds to a point-wise partitioning of \( QO \) for which \( Q_{DU(H)} = Q_{DL(H)} = 0 \). Hence, for iterative methods based on block partitionings \( l = 1, 2, \ldots, H-1 \) should be used.

### 4.2. Example (continued).

Consider the block partitioning of the 4-dimensional problem at level 1 for which \( b_1 = 4 \), and \( Q \) is viewed as a \((2 \times 2)\) block matrix with blocks of order \( o_1 = 8 \) (see (2.2)). Then, from (4.3) and (4.4), we have

\[ Q_{U(1)} = \sum_{k=1}^{7} U_{k}^{(1)} \otimes Q_{k}^{(2)} \otimes Q_{k}^{(3)} \otimes Q_{k}^{(4)} \] and \( Q_{L(1)} = \sum_{k=1}^{7} L_{k}^{(1)} \otimes Q_{k}^{(2)} \otimes Q_{k}^{(3)} \otimes Q_{k}^{(4)} \),

implying \( Q_{U(1)} + Q_{L(1)} = \) \begin{pmatrix} 1 & 1 & 1 & 1 \\ 10 & 10 & 10 & 10 \end{pmatrix}.

Whereas from (4.5) and (4.6), we have

\[ Q_{DU(1)} = \sum_{k=1}^{7} D_{k}^{(1)} \otimes U_{k}^{(2)} \otimes Q_{k}^{(3)} \otimes Q_{k}^{(4)} + \sum_{k=1}^{7} D_{k}^{(1)} \otimes D_{k}^{(2)} \otimes U_{k}^{(3)} \otimes Q_{k}^{(4)} + \sum_{k=1}^{7} D_{k}^{(1)} \otimes D_{k}^{(2)} \otimes D_{k}^{(3)} \otimes U_{k}^{(4)} \]

\[ Q_{DL(1)} = \sum_{k=1}^{7} D_{k}^{(1)} \otimes L_{k}^{(2)} \otimes Q_{k}^{(3)} \otimes Q_{k}^{(4)} + \sum_{k=1}^{7} D_{k}^{(1)} \otimes D_{k}^{(2)} \otimes L_{k}^{(3)} \otimes Q_{k}^{(4)} + \sum_{k=1}^{7} D_{k}^{(1)} \otimes D_{k}^{(2)} \otimes D_{k}^{(3)} \otimes L_{k}^{(4)} \]

implying \( Q_{DU(1)} + Q_{DL(1)} = \) \begin{pmatrix} 1 & 1 & 1 & 1 \\ 10 & 10 & 10 & 10 \\ 1 & 1 & 1 & 1 \\ 10 & 10 & 10 & 10 \end{pmatrix}.
Note that there are the $\sqrt{n} = 2$ blocks along the diagonal.

4.3. Block iterative methods for Kronecker products. Now, let $Q$ in (2.1) be irreducible and split at level $l$ using (4.2) as

\begin{equation}
Q = Q_O + Q_D = Q_{U(l)} + Q_{L(l)} + Q_{DU(l)} + Q_{DL(l)} + Q_D = M - N,
\end{equation}

where $M$ is nonsingular (i.e., $M^{-1}$ exists). Then, the power, block Jacobi over-relaxation (BJOR), and block successive over-relaxation (BSOR) methods are based on different splittings of $Q$, and each satisfies

$$
\pi_{(m+1)}M = \pi_{(m)}N \quad \text{for} \quad m = 0, 1, \ldots
$$

with the sequence of approximations $\pi_{(m+1)}$ to the steady-state distribution in (1.2), where $\pi(0) > 0$ is the initial approximation such that $\pi(0)e = 1$ and $T = NM^{-1}$ is the iteration matrix. Note that $T$ does not change from iteration to iteration and only the current approximation is used to compute the new approximation. Hence, these methods, which are based on splittings of the coefficient matrix, are also known as stationary iterative methods. Since $Q$ is a singular matrix and assumed to be irreducible, the largest eigenvalue of $T$ in magnitude is one. In order to ensure convergence, $T$ should not have other eigenvalues with magnitude one. For converging approximations, the magnitude of the eigenvalue of $T$ closest to one determines the rate of convergence [62].

The particular splittings corresponding to the power, BJOR, and (forward) BSOR methods are

$$
M^{\text{Power}} = -\alpha I,
$$

$$
N^{\text{Power}} = -\alpha (I + Q/\alpha),
$$

$$
M^{\text{BJOR}} = (Q_D + Q_{DU(l)} + Q_{DL(l)})/\omega,
$$

$$
N^{\text{BJOR}} = (1 - \omega)(Q_D + Q_{DU(l)} + Q_{DL(l)})/\omega - Q_{U(l)} - Q_{L(l)},
$$

$$
M^{\text{BSOR}} = (Q_D + Q_{DU(l)} + Q_{DL(l)})/\omega + Q_{U(l)},
$$

$$
N^{\text{BSOR}} = (1 - \omega)(Q_D + Q_{DU(l)} + Q_{DL(l)})/\omega - Q_{L(l)},
$$

where $\alpha \in [\max_{s \in S} |q_D(s, s)|, \infty)$ is the uniformization parameter of the power method and $\omega \in (0, 2)$ is the relaxation parameter of the BJOR and BSOR methods. The power method works at level $l = H$ since it is a point method. Furthermore, the BJOR and BSOR methods reduce to the block Jacobi (BJacobi) and block Gauss-Seidel (BGS) methods for $\omega = 1$, and the BJOR and BSOR methods become point JOR and point SOR methods for $l = H$. We remark that [40] shows how one can find $\max_{s \in S} |q_D(s, s)|$ in the presence of functional transitions when $Q_D$ is given as a sum of Kronecker products.

Since $Q = Q_O + Q_D$, it is possible to express the power method at iteration $m$ as

\begin{equation}
\pi_{(m+1)} = \pi_{(m)} + \pi_{(m)}Q_D/\alpha + \pi_{(m)}Q_O/\alpha.
\end{equation}

Observe that the second term in (4.8) poses no problem from a computational point of view since $Q_D$ is diagonal, and the third term can be efficiently implemented using the vector-Kronecker product multiplication algorithm since $Q_O$ is a sum of Kronecker products (see (2.1)).

Regarding the BJOR method with a level $l$ block partitioning, at iteration $m$ we have

$$
\pi_{(m+1)}(Q_D + Q_{DU(l)} + Q_{DL(l)})
$$
(4.9) \[ (1 - \omega)\pi(m)Q_D + (1 - \omega)\pi(m)Q_{DU(l)} + (1 - \omega)\pi(m)Q_{DL(l)} - \omega\pi(m)Q_{U(l)} - \omega\pi(m)Q_{L(l)}. \]

This is a block diagonal linear system with \( \sqrt{l} \) blocks of order \( o_l \) along the diagonal of the nonsingular coefficient matrix \((Q_D + Q_{DU(l)} + Q_{DL(l)})\) and a nonzero right-hand side which can be efficiently computed using the vector-Kronecker product multiplication algorithm, since \(Q_{U(l)}\), \(Q_{L(l)}\), \(Q_{DU(l)}\), and \(Q_{DL(l)}\) are sums of Kronecker products (see (4.3), (4.4), (4.5), and (4.6)). Hence, (4.9) is equivalent to \( \sqrt{l} \) independent, nonsingular linear systems each of order \( o_l \) and a nonzero right-hand side. If there is space, one can generate and factorize in sparse storage the nonsingular blocks of the form

\[ Q((i_1, i_2, \ldots, i_t), (i_1, i_2, \ldots, i_t)) = \sum_{k=1}^{K} \left( \prod_{h=1}^{l} q_k^{(h)}(i_h, i_h) \right) \left( H_{h=t+1}^{(h)} Q_k^{(h)} \right) \]

(4.10) \[ + Q_D((i_1, i_2, \ldots, i_t), (i_1, i_2, \ldots, i_t)) \text{ for } (i_1, i_2, \ldots, i_t) \in \times_{h=1}^{l} S^{(h)} \]

along the diagonal (see (2.3)) of \((Q_D + Q_{DU(l)} + Q_{DL(l)})\) at the outset and solve the \(| \times_{h=1}^{l} S^{(h)} | = \sqrt{l} \) systems directly at each iteration. Otherwise, one can use an iterative method; indeed, it is even possible to use a block iterative method, such as BSOR, since the off-diagonal parts of the diagonal blocks given by

\[ \sum_{k=1}^{K} \left( \prod_{h=1}^{l} q_k^{(h)}(i_h, i_h) \right) \left( H_{h=t+1}^{(h)} Q_k^{(h)} \right) \]

are sums of Kronecker products.

The situation with the BSOR method is not very different from that of BJOR. For BSOR with a level \( l \) block partitioning, at iteration \( m \) we have

\[ \pi_{m+1}(Q_D + Q_{DU(l)} + Q_{DL(l)} + \omega Q_{U(l)}) \]

(4.11) \[ = (1 - \omega)\pi(m)Q_D + (1 - \omega)\pi(m)Q_{DU(l)} + (1 - \omega)\pi(m)Q_{DL(l)} - \omega\pi(m)Q_{L(l)}. \]

This is a block upper-triangular linear system with \( \sqrt{l} \) blocks of order \( o_l \) along the diagonal of the nonsingular coefficient matrix \((Q_D + Q_{DU(l)} + Q_{DL(l)} + \omega Q_{U(l)})\) and a nonzero right-hand side which can be efficiently computed using the vector-Kronecker product multiplication algorithm, since \(Q_{U(l)}\), \(Q_{DU(l)}\), and \(Q_{DL(l)}\) are sums of Kronecker products. In [64], a recursive algorithm is given for a nonsingular linear system with a lower-triangular coefficient matrix in the form of a sum of Kronecker products and a nonzero right-hand side. Such a system arises in backward point SOR. Therein, a version of the same algorithm for backward BSOR is also discussed. Here we remark that an iterative block upper-triangular solution algorithm for (4.11) is also possible [18] and a block row-oriented version is preferable in the presence of functional transitions:
Algorithm 1. Iterative block upper-triangular solution at level \( l \) for MCs based on Kronecker products

\[
b = (1 - \omega)\pi^{(m)}Q_D + (1 - \omega)\pi^{(m)}Q_{DU(l)} + (1 - \omega)\pi^{(m)}Q_{DL(l)} - \omega\pi^{(m)}Q_L(l);
\]

For row of blocks \((i_1, i_2, \ldots, i_l) = (1, 1, \ldots, 1)\) to \((n_1, n_2, \ldots, n_l)\) lexicographically,

- Solve \(\pi^{(m+1)}((i_1, i_2, \ldots, i_l))Q((i_1, i_2, \ldots, i_l), (i_1, i_2, \ldots, i_l)) = b(i_1, i_2, \ldots, i_l);\)
- For column of blocks \((j_1, j_2, \ldots, j_l) > (i_1, i_2, \ldots, i_l), b((j_1, j_2, \ldots, j_l)) = b((j_1, j_2, \ldots, j_l)) - \omega\pi^{(m+1)}((i_1, i_2, \ldots, i_l))Q_{U(l)}((i_1, i_2, \ldots, i_l), (j_1, j_2, \ldots, j_l)).\)

Observe in Algorithm 1 that initially the nonzero right-hand side vector \(b\) can be efficiently computed using the vector-Kronecker product multiplication algorithm, since \(Q_L(l), Q_{DU(l)},\) and \(Q_{DL(l)}\) are sums of Kronecker products. Furthermore, \(Q((i_1, i_2, \ldots, i_l), (i_1, i_2, \ldots, i_l))\) is given in (4.10) in terms of a sum of Kronecker products, and \(Q_{U(l)}((i_1, i_2, \ldots, i_l), (j_1, j_2, \ldots, j_l))\) for \((j_1, j_2, \ldots, j_l) > (i_1, i_2, \ldots, i_l)\) can be expressed in terms of a sum of Kronecker products using (4.3) as

\[
Q((i_1, i_2, \ldots, i_l), (j_1, j_2, \ldots, j_l)) = \sum_{k=1}^{K} \sum_{l=1}^{L} \left( \prod_{j=1}^{l} d_k^f(i_f, j_f) \right) u_k^{(b)}(i_h, j_h) \left( \prod_{f=h+1}^{l} q_k^f(i_f, j_f) \right) \left( \bigotimes_{f=h+1}^{l} Q_k^f \right).
\]

To the contrary of BJOR, the nonsingular diagonal blocks \(Q((i_1, i_2, \ldots, i_l), (i_1, i_2, \ldots, i_l))\) in BSOR must be solved in lexicographical order. If there is space, one can generate and factorize in sparse storage these blocks as in BJOR at the outset and solve the \(\sqrt{b_l}\) systems directly at each iteration. Otherwise, one can use an iterative method such as BSOR, since the off-diagonal parts of the diagonal blocks are also sums of Kronecker products. After each block is solved for the unknown subvector \(\pi^{(m+1)}((i_1, i_2, \ldots, i_l))\), \(b\) is updated by multiplying the computed subvector with the corresponding row of blocks above the diagonal. Finally, we emphasize that BSOR at level \(l\) reduces to point SOR if \(Q_{DL(l)} = 0\) (see Remark 4.1 in [64, p. 176]).

It is quite surprising to notice that block iterative solvers, which are sometimes called two-level iterative solvers, have still not been incorporated into most analysis packages based on Kronecker representations although they are shown to be more effective than point solvers on many test cases [18, 64]. Furthermore, to the contrary of block partitionings considered in [33] for sparse MCs, block partitionings of Kronecker products are nested and recursive due to the lexicographical ordering of states. Therefore, there tends to be more common structure among the diagonal blocks of a MC expressed as a sum of Kronecker products. Diagonal blocks having identical off-diagonal parts and diagonals which differ by a multiple of the identity are exploited in [18]. Therein, it is shown that such diagonal blocks can share and work with the factorization of only one diagonal block. This approach saves not only from time spent for factorization of diagonal blocks at the outset, but also from space. The same paper also discusses a three-level version of BSOR for MCs based on Kronecker products in which the diagonal blocks that are too large to be factorized are solved using BSOR. Similar results also appear in [39]. Finally, we remark that it is possible to alter the nonzero structure of the underlying MC of a Kronecker representation by reordering factors and states of factors so as to make it more suitable for block iterative methods. Obviously, the power and point JOR methods will not benefit from such reordering.
In the next section, we introduce a simple version of the multilevel (ML) method [15, 19] for irreducible MCs based on Kronecker products which happens to be a generalization of the well-known iterative aggregation-disaggregation (IAD) method [62] to more than two levels.

5. Multilevel methods. In [6, 11, 12, 13], aggregation-disaggregation steps are coupled with various iterative methods for MCs based on Kronecker products to accelerate convergence. An IAD method for MCs based on Kronecker products and its adaptive version, which analyzes aggregated systems for those parts where the error is estimated to be high, are proposed in [9] and [14], respectively. The adaptive IAD method in [14] is improved in [15] through a recursive definition and called ML.

5.1. The simple multilevel method for Kronecker products. Let \( S(0) = H_{0=0}^H S(h) \) for \( l = 0, 1, \ldots, H \) and the mapping \( f(l) : S(l) \rightarrow S(l+1) \) represent the aggregation of dimension \( (l+1) \) (i.e., the state space \( S(l+1) \)) so that the states in \( S(l) \) are mapped to the states in \( S(l+1) \). Note that \( S(0) = S \) and \( S(H) = \{1 \} \). Furthermore, let the aggregated CTMCs \( \hat{Q}(m,l) \) with state spaces \( S(l) \) be defined at levels \( l = 1, 2, \ldots, H \) with \( \hat{Q}(m,0) = Q \) for iteration \( m \). Finally, let the power method be used as a smoother (or accelerator) before aggregation \( \eta(m,l) \) times and after disaggregation \( \nu(m,l) \) times with \( \alpha(m,l) \in \max_{s(t) \in S(l)} [\hat{Q}(m,l)(s(t)), s(t)] \) at level \( l \) for iteration \( m \). Then the ML iteration matrix at level \( l \) for iteration \( m \) is given by

\[
T_{(m,l)}^{ML} = (I + \hat{Q}(m,l)/\alpha(m,l))^{\eta(m,l)} R(l) T_{(m,l+1)}^{ML} P_{x(m,l)} (I + \hat{Q}(m,l)/\alpha(m,l))^{\nu(m,l)},
\]

and satisfies

\[
\pi_{(m+1,l)} = \pi_{(m,l)} T_{(m,l)}^{ML} \quad \text{for} \quad m = 0, 1, \ldots,
\]

where

\[
x_{(m,l)} = \pi_{(m,l)} (I + \hat{Q}(m,l)/\alpha(m,l))^{\eta(m,l)},
\]

\[
r(l)(s(t), s(t+1)) = \begin{cases} 
1 & \text{if } f(l)(s(t)) = s(t+1) \\
0 & \text{otherwise}
\end{cases}
\]

\[
for \quad s(t) \in S(l) \quad \text{and} \quad s(t+1) \in S(l+1),
\]

\[
p_{x(m,l)} (s(t+1), s(t)) = \begin{cases} 
\frac{x_{(m,l)}(s(t))}{\sum_{s(t) \in S(l)} x_{(m,l)}(s(t))} & \text{if } f(l)(s(t)) = s(t+1) \\
0 & \text{otherwise}
\end{cases}
\]

\[
for \quad s(t+1) \in S(l+1) \quad \text{and} \quad s(t) \in S(l),
\]

\[
\pi_{(m+1,l)} = x_{(m,l)} R(l) \quad \text{and} \quad \hat{Q}_{(m,l+1)} = P_{x(m,l)} \hat{Q}_{(m,l)} R(l).
\]

At iteration \( m \), the recursion ends and backtracking starts when \( \hat{Q}_{(m,l+1)} \) in (5.5) is the last aggregated CTMC and solved exactly to give \( T_{(m,l+1)} = \epsilon \pi_{(m+1,l+1)} \), where \( \pi_{(m+1,l+1)} \hat{Q}_{(m,l+1)} = 0 \) and \( \pi_{(m+1,l+1)} \epsilon = 1 \). The level to end recursion depends on available memory since there must be space to store and factorize the aggregated CTMC at that level. When the initial approximation is positive (i.e., \( \pi_{(0,0)} > 0 \),
the aggregated CTMCs $\tilde{Q}_{(m,t+1)}$ are irreducible [15, p. 348], and the ML method has been observed to converge if a sufficient number of smoothings are performed to improve the approximate solution vector, $\pi_{(m,t)}$, at each level.

Observe that to the contrary of block iterative methods, the ML iteration matrix in (5.1) changes from iteration to iteration, and hence, the method is non-stationary. Nevertheless, the $(|S(t)| \times |S(t+1)|)$ aggregation operator, $R_{(t)}$, in (5.3) is constant and need not be stored since it is defined by $f_{(l)}$. At level $l$, the $|S(t)| = \prod_{h=1}^{H} n_l$ states represented by $(H - l)$-tuples are mapped to the $|S(t+1)| = \prod_{h=1}^{H} n_l$ states represented by $(H - l - 1)$-tuples by aggregating the leading dimension $S^{(l+1)}$ in $S(t)$. We remark that this corresponds to an aggregation based on a contiguous and non-interleaved block partitioning if the states in $S(t)$ were ordered anti-lexicographically. On the other hand, the $(|S_{(t+1)}| \times |S_{(t)}|)$ disaggregation operator, $P_{x(m,t)}$, in (5.4) depends on the smoothed vector $x_{(m,t)}$ in (5.2) and has the nonzero structure of $R_T^{(l)}$. Therefore, $P_{x(m,t)}$ can be stored in a vector of length $|S(t)|$ since it has one nonzero per column by definition. These vectors amount to a total storage of $\sum_{h=0}^{H-1} \prod_{h=1}^{H} n_h$ floating-point values if the recursion terminates at level $H$.

In [15, p. 347], it is shown that $\tilde{Q}_{(m,t+1)}$ can be expressed as a sum of Kronecker products using at most $K$ vectors of length $|S(t)|$ and the matrices corresponding to the factors $(l + 2)$ through $H$. More specifically, the $s_{(t+1)}$st element of the vector corresponding to the $k$th term in the Kronecker representation at level $(l+1)$ for iteration $m$ is defined as

$$a_{(m,t+1),k}(s_{(t+1)}) = \frac{\sum_{s(t) \in S(t), f_{(l)}(s(t)) = s_{(t+1)}} x_{(m,t)}(s_{(l)}) a_{(m,t),k}(s_{(l)}) (e_{s_{(l)}}(l+1)^T Q_k^{(l+1)} e)}{\pi_{(m,t+1)}(s_{(t+1)})}$$

for $s_{(t+1)} \in S_{(t+1)}$ and $k = 1, 2, \ldots, K$,

where $a_{(m,0),k} = e$, $s_{(l+1)}(l+1) \in S^{(l+1)}$, and $e_{s_{(l)}}(l+1)$ is the $s_{(l)}$th column of $I$.

Then

$$\tilde{Q}_{(m,t+1)} = \sum_{k=1}^{K} \text{diag}(a_{(m,t+1),k}) \bigotimes_{h=1}^{H} Q_k^{(h)}$$

(5.7)

$$- \sum_{k=1}^{K} \text{diag}(a_{(m,t+1),k}) \bigotimes_{h=1}^{H} \text{diag}(Q_k^{(h)} e).$$

Observe that the second summation in (5.7) returns a diagonal matrix which sums the rows of $\tilde{Q}_{(m,t+1)}$ to zero. Furthermore, the vectors $a_{(m,0),k}$ for $k = 1, 2, \ldots, K$ at level 0 consist of all ones, and therefore need not be stored. If the recursion ends at level $H$, then $\tilde{Q}_{(m,H)}$ is a $(1 \times 1)$ CTMC equal to zero, and need not be stored since its steady-state vector is one. We remark that $a_{(m,t+1),k} = e$ for those $k$ which either have a single $Q_k^{(h)} \neq I$ for $h = 1, 2, \ldots, H$, or have all $Q_k^{(h)} = I$ for $h = l + 2, \ldots, H$. Such vectors need not be stored either. The $K$ vectors at a particular level have the same length, but vary in length from $\prod_{h=2}^{H} n_h$ at level 1 to $n_H$ at level $(H - 1)$, implying a storage requirement of at most $K \sum_{h=1}^{H-1} \prod_{h=1}^{H} n_h$ floating-point values to facilitate the Kronecker representation of the aggregated CTMCs. We remark that grouping of factors will further reduce the storage requirement for vectors.
5.2. Example (continued). Consider the 4-dimensional problem with the initial distribution \( \pi(0,0) = e/16 \), \( \alpha(0,0) = 21 \), and \( \eta(0,0) = \nu(0,0) = 1 \). Then, \( x(0,0) = \pi(0,0)(I + \tilde{Q}(0,0)/21) \) from (5.2) yields
\[
x(0,0) = (19 11 21 13 27 19 29 21 13 23 15 29 21 31 23)/336.
\]
Furthermore,
\[
R(0) = \begin{pmatrix}
1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1
\end{pmatrix}
\]
and
\[
P_{x(0,0)} = \begin{pmatrix}
\frac{19}{336} & & & \\
\frac{21}{336} & & & \\
\frac{13}{336} & \frac{27}{336} & & \\
\frac{21}{336} & \frac{29}{336} & \frac{15}{336} & \\
\frac{19}{336} & \frac{22}{336} & \frac{21}{336} & \frac{31}{336} \\
\frac{21}{336} & \frac{44}{336} & \frac{24}{336} & \frac{34}{336}
\end{pmatrix}
\]
from (5.3) and (5.4), respectively. Hence, the 16 states represented by 4-tuples in \( S(0) = S \) are mapped to the 8 states represented by 3-tuples in \( S(1) \). For instance, states \( (1,1,1,1) \) and \( (2,1,1,1) \) are mapped to \( (1,1,1) \), whereas states \( (1,1,1,2) \) and \( (2,1,1,2) \) are mapped to \( (1,1,2) \). Using \( R(0) \) in (5.5), we obtain the starting approximation at level 1 as
\[
\pi(0,1) = (40 24 44 28 56 40 60 44)/336.
\]
Through (5.6), the 7 vectors used to represent the aggregated CTMC at level 1 are computed as
\[
a_{(0,1),1} = (19/40 11/24 21/44 13/28 27/56 19/40 29/60 21/44),
a_{(0,1),2} = a_{(0,1),3} = a_{(0,1),4} = a_{(0,1),5} = e,
a_{(0,1),6} = (19/40 11/24 21/44 13/28 27/56 19/40 29/60 21/44),
a_{(0,1),7} = (210/44 130/24 230/44 150/28 290/56 210/40 310/60 230/44).
\]
and the aggregated CTMC is expressed as

\[ \tilde{Q}_{0,1} = P_{x(0,0)} \tilde{Q}_{(0,0)} R(0) \]

\[ = \sum_{k=1}^{7} \text{diag}(a(0,1,k)) \bigotimes_{h=2}^{4} Q_k^{(h)} - \sum_{k=1}^{7} \text{diag}(a(0,1,k)) \bigotimes_{h=2}^{4} \text{diag}(Q_k^{(h)} e). \]

Observe that the effect of \(a(0,1,1)\) in the first term of the first summation will be to the diagonal of \(\tilde{Q}_{0,1}\) since \(Q_1^{(2)} = Q_1^{(3)} = Q_1^{(4)} = I\). But this effect will be cancelled by the first term of the second summation simply because \(\text{diag}(Q_1^{(2)} e) = \text{diag}(Q_1^{(3)} e) = \text{diag}(Q_1^{(4)} e) = I\). Hence, we may very well set \(a(0,1,1) = e\) as suggested before.

Furthermore, \(a(0,1,2) = a(0,1,3) = a(0,1,4) = a(0,1,5) = e\) since \(Q_2^{(1)} = Q_3^{(1)} = Q_4^{(1)} = Q_5^{(1)} = I\) and \(a(0,0,k) = e\) for \(k = 1, 2, \ldots, 7\). Therefore, we implicitly have

\[
\begin{pmatrix}
1 & 1 & 1 & 1 & 2 & 2 & 2 \\
1 & 1 & 2 & 2 & 1 & 1 & 2 \\
1 & 2 & 1 & 2 & 1 & 2 & 1 \\
\end{pmatrix}
\]

\[
\tilde{Q}_{(0,1)} = \begin{pmatrix}
290 & 40 & 40 & 210 \\
-394 & 240 & 24 & 130 \\
210 & 244 & 24 & 240 \\
130 & 240 & 150 \\
56 & -168 & 56 & 56 \\
40 & -480 & 400 & 40 \\
60 & 290 & 210 & -254
\end{pmatrix}
\]

In the next step, similar operations will be carried out at level \(1\) unless the aggregated CTMC is solved exactly, upon which backtracking from recursion starts for iteration \(m\).

5.3. A class of multilevel methods for Kronecker products. The ML method we discussed follows a V-cycle [48] at each iteration. That is, starting from the finest level, at each step it smoothes the current approximation and moves to a coarser level by aggregation until it reaches a level at which the aggregated CTMC can be solved exactly. Once the exact solution is obtained at the coarsest level, the method starts moving in the opposite direction. At each step on the way to the finest level, the method disaggregates the current approximation passed by the coarser level and smoothes it. Furthermore, the state spaces \(S^{(h)}\) are aggregated according to the fixed ordering \(h = 1, 2, \ldots, H\). However, to the contrary of the ML method for sparse MCs in [45], the definition of the aggregated state spaces follows naturally from the Kronecker representation in (2.1) and the aggregated CTMCs can also be represented using Kronecker products as shown in (5.7). In [19], a sophisticated class of ML methods is given. The methods therein are capable of using JOR and SOR as smoothers, performing the W- and F-cycles inspired by multigrid [67], and aggregating the state spaces in cyclic and adaptive orderings. Numerical experiments in [19] proved ML methods to be very strong, robust, and scalable solvers for MCs based on Kronecker products.

The convergence properties of the class of ML methods in [19] are discussed in [21]; however, it is not clear how its behavior would be affected if block iterative
methods, such as BJOR and BSOR, are used as smoothers rather than power, JOR, and SOR. Note that BJOR and BSOR should normally not use a direct method for the solution of the diagonal blocks when employed as smoothers with the ML method, since the aggregated CTMC at each level changes from iteration to iteration and the factorization may be too time consuming to offset. In [40], an efficient algorithm that finds a nearly completely decomposable (NCD) [33, 62] partitioning of $S$ in the presence of functional transitions for a user specified decomposability parameter is given. Since IAD using NCD partitionings has certain rate of convergence guarantees, the algorithm may be useful in the context of ML methods to determine the loosely coupled dimensions to be aggregated first in a given iteration.

The next section discusses various preconditioners to be used with projection methods for MCs based on Kronecker products.

6. Preconditioned projection methods. Projection (or Krylov subspace) methods for MCs based on Kronecker products are non-stationary iterative methods in which approximate solutions satisfying various constraints are extracted from small dimensional subspaces [62]. Being iterative, their basic operation is vector-Kronecker product multiplication. However, compared to block iterative methods, they require a larger number of supplementary vectors as long as the state space size, $n$. But, more importantly, they need to be used with preconditioners to result in effective solvers.

At each iteration of a preconditioned projection method, the row residual vector, $r$, is used as the right-hand side of the linear system

$$zM = r \quad (6.1)$$

to compute the preconditioned row residual vector, $z$. The objective of this preconditioning step is to improve the error in the approximate solution vector at that iteration. Note that if $M$ were a multiple of $I$ (as in (4.8)), the preconditioned residual would be a multiple of the residual computed at that iteration, implying no improvement. Hence, the preconditioner should approximate the coefficient matrix of the original system in a better way, yet the solution of linear systems as in (6.1) involving the preconditioner matrix, $M$, should be cheap. It is shown in [33] through a large number of numerical experiments on benchmark problems that, to result as effective solvers, projection methods for sparse MCs should be used with preconditioners, such as those based on incomplete LU (ILU) factorizations. However, it is still not clear how one can devise ILU-type preconditioners for MCs that are in the form of (2.1).

So far, various preconditioners are proposed for Kronecker structured representations such as those based on truncated Neumann series [62, 63], the cheap and separable preconditioner [12], and circulant preconditioners for a specific class of problems [25]. The Kronecker product approximate preconditioner for MCs based on Kronecker products developed in a sequence of papers [49, 50, 51], although encouraging, is in the form of a prototype implementation. Numerical experiments in [12, 13, 50, 51, 63] indicate that there is still room for research regarding the development of effective preconditioners for MCs based on Kronecker products.

In introducing another class of preconditioners, we remark that each of the block iterative methods introduced in this work is actually a preconditioned power method for which the preconditioning matrix is $M$ in (4.7). Since $M$ is based on Kronecker products, a BSOR preconditioner exploiting this property is proposed in [20]. To the contrary of the BSOR preconditioner entertained for sparse MCs in [33], the BSOR preconditioner for MCs based on Kronecker products has a rich structure induced by the lexicographical ordering of states. Through numerical experiments, it is shown in
that two-level BSOR preconditioned projection methods in which the diagonal blocks are solved exactly emerge as effective solvers that are competitive with block iterative methods and ML methods.

It will be interesting to compare point JOR, BJOR, and point SOR preconditioners as defined in (4.9) and (4.11) with the existing preconditioners for MCs based on Kronecker products. Clearly, the class of ML methods proposed in [19] is another candidate for preconditioning projection methods.

7. Conclusion. MCs based on Kronecker products have a rich structure, which is nested and recursive. Preprocessing techniques that take advantage of this rich structure to expedite analysis are reordering, grouping, and lumping. Block iterative methods based on splittings, multilevel methods, and projection methods preconditioned with block iterative methods come across as a strong set of solvers which should be integrated to software packages working with Kronecker products. However, all of this is easier said, than done. Implementation of these solvers requires intricate programming with dynamically allocated, relatively complex data structures, which needs time, careful testing, and tuning.

REFERENCES

T. DAYAR


[32] T. Dayar, Effects of reordering and lumping in the analysis of discrete-time SANs, in


