

Chapter 2

Preliminaries

The *Kronecker* (or *tensor*) *product* [50, 148] of two (rectangular) matrices A and B with $A = [a(i_A, j_A)]$ is

$$A \otimes B = [a(i_A, j_A)B].$$

Or, more formally, given $A \in \mathbb{R}^{n_A \times m_A}$ and $B \in \mathbb{R}^{n_B \times m_B}$, $A \otimes B$ yields the (rectangular) matrix $C \in \mathbb{R}^{n_A n_B \times m_A m_B}$ whose entries satisfy

$$c(i_C, j_C) = a(i_A, j_A)b(i_B, j_B),$$

$$\text{with } i_C = i_A n_B + i_B \text{ and } j_C = j_A m_B + j_B$$

for

$$(i_A, j_A) \in \{0, \dots, n_A - 1\} \times \{0, \dots, m_A - 1\},$$

$$(i_B, j_B) \in \{0, \dots, n_B - 1\} \times \{0, \dots, m_B - 1\},$$

where $\times$ is the *Cartesian product* operator. Note that in a two-dimensional representation, the row indices of C are in $\{0, \dots, n_A - 1\} \times \{0, \dots, n_B - 1\}$, whereas its column indices are in $\{0, \dots, m_A - 1\} \times \{0, \dots, m_B - 1\}$. Hence, the ordering of rows and columns of C with respect to this two-dimensional representation is *lexicographical* since

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

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

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

where $X^{(h)} \in \mathbb{R}^{n_h \times n_h}$ and row/column indices of $X^{(h)}$ are in $\mathcal{S}^{(h)} = \{0, \dots, n_h - 1\}$ for $h = 1, \dots, H$. Therefore, $X \in \mathbb{R}^{n \times n}$, with $n = \prod_{h=1}^H n_h$, and the ordered H -dimensional tuples

$$\mathbf{i} = (i_1, \dots, i_H) \in \times_{h=1}^H \mathcal{S}^{(h)} \quad \text{and} \quad \mathbf{j} = (j_1, \dots, j_H) \in \times_{h=1}^H \mathcal{S}^{(h)}$$

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 (or *flat*) state space that are lexicographically ordered, and naturally the Kronecker product has been used to define MCs having multidimensional state spaces.

2.1 Kronecker Representation

We assume that the infinitesimal generator of the H -dimensional CTMC at hand is represented as a sum of Kronecker products plus a diagonal matrix. Specifically, we have

$$Q = Q_O + Q_D, \quad Q_O = \sum_{k=1}^K \bigotimes_{h=1}^H Q_k^{(h)}, \quad Q_D = -\text{diag}(Q_O \mathbf{e}), \quad (2.1)$$

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, $Q_k^{(h)} \in \mathbb{R}_{\geq 0}^{n_h \times n_h}$ for $k = 1, \dots, K$ and $h = 1, \dots, H$. Observe that $Q_O \geq 0$ and $Q_D \leq 0$. If row/column indices of $Q_k^{(h)}$ are in $\mathcal{S}^{(h)} = \{0, \dots, n_h - 1\}$ for $k = 1, \dots, K$ and $h = 1, \dots, H$, then the H -dimensional state space of Q is given by

$$\mathcal{S} = \times_{h=1}^H \mathcal{S}^{(h)},$$

the *product state space*. Observe that $|\mathcal{S}| = \prod_{h=1}^H |\mathcal{S}^{(h)}| = \prod_{h=1}^H n_h = n$. Furthermore, the one-dimensional value of state $\mathbf{i} = (i_1, \dots, i_H) \in \mathcal{S}$, where $i_h \in \mathcal{S}^{(h)}$ for $h = 1, \dots, H$, is given by

$$i = \sum_{h=1}^H i_h \prod_{l=h+1}^H n_l.$$

Oftentimes, when implementing an algorithm, one needs to have a mapping from the one-dimensional state space to the multidimensional state space, and vice versa, since solution vectors are one-dimensional, and appropriate entries of them need to be accessed. Therefore, we will be using the one-dimensional and multidimensional representations of states interchangeably throughout the text.

Each factor in this representation is associated with a different subsystem, and there are H of them. Without loss of generality, we assume that the first H of the K Kronecker product terms model *local* transitions, the k th one being that of subsystem k , whereas the remaining $(K - H)$ Kronecker product terms model *synchronized* transitions. In each of the first K terms, there is only one factor different than the identity matrix; for the k th term it is the k th factor. In each of the remaining $(K - H)$ terms, there are at least two factors different than the identity matrix, and those correspond to subsystems that get synchronized by that particular transition. The existence of an identity matrix in a term implies that the corresponding subsystem does not change its state during that particular transition. Hence, only one subsystem may change its state during a local transition, and at least two subsystems may change their states during a synchronized transition. Note that it is also possible to represent the sum of the first K terms of Kronecker products as a single term of *Kronecker sum* and express the remaining $(K - H)$ terms as a sum of Kronecker products. This has been done for the SAN and HMM formalisms. It is our opinion that this complicates the already cumbersome notation further, and we therefore refrain from doing that here.

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 $h = 1, \dots, H$. In the worst case, this amounts to a storage space of $n + \sum_{h=1}^H nz_{Q^{(h)}} n_h$ floating-point values, where $nz_{Q^{(h)}}$ is the sum of the number of nonzeros in $Q_k^{(h)}$ for $k = 1, \dots, K$, compared to nz nonzeros required by the flat representation. Q_D can also be expressed as a sum of Kronecker products:

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

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

Now, each nonzero entry of matrix $Q_k^{(h)}$ in (2.1) is located by its row and column indices, which are members of $\mathcal{S}^{(h)}$. In a more general setting, a nonzero entry of $Q_k^{(h)}$ may be a function of states in state spaces other than $\mathcal{S}^{(h)}$ and, thus, a function of nonlocal states. This phenomenon is a byproduct of the modeling process and is utilized in the SAN modeling formalism. These nonzero entries are referred to as *functional transitions*, and the corresponding Kronecker products are said to be *generalized* [119]. Although it is possible to remove functional transitions from a sum of generalized Kronecker products by introducing new terms [121] or factors [10], functional transitions enable a more compact Kronecker representation with denser factor matrices [42]. We do not consider functional transitions in this discussion but indicate that the results extend to generalized Kronecker products wherever appropriate.

There is a rich structure associated with the Kronecker representation in (2.1). This structure is nested and recursive [32–34, 54, 85, 87, 88, 146]. Let level 0 denote

the highest level at which Q is perceived as a single block of order $n = \prod_{h=1}^H n_h$. At the next level, which we call level 1, Q is an $(n_1 \times n_1)$ block matrix with blocks of order $\prod_{h=2}^H n_h$. At level 2, Q is an $(n_1 n_2 \times n_1 n_2)$ block matrix with blocks of order $\prod_{h=3}^H n_h$. 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

$$b_l = \begin{cases} 1, & l = 0 \\ n_l^2 b_{l-1}, & l = 1, \dots, H \end{cases} \quad \text{and} \quad o_l = \begin{cases} n, & l = 0 \\ o_{l-1}/n_l, & l = 1, \dots, H \end{cases}, \quad (2.2)$$

where b_l and o_l denote the number and order of blocks at level $l = 0, 1, \dots, H$, respectively. Unrolling the recurrences, we obtain

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

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

$$\begin{aligned} & Q((i_1, \dots, i_l), (j_1, \dots, j_l)) \\ &= \sum_{k=1}^K \left(\prod_{h=1}^l q_k^{(h)}(i_h, j_h) \right) \left(\bigotimes_{h=l+1}^H Q_k^{(h)} \right) + Q_D((i_1, \dots, i_l), (j_1, \dots, j_l)) \quad (2.3) \\ & \quad \text{for } l = 0, \dots, H, \end{aligned}$$

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

$$\begin{aligned} q((i_1, \dots, i_H), (j_1, \dots, j_H)) &= \sum_{k=1}^K \prod_{h=1}^H q_k^{(h)}(i_h, j_h) \\ &+ q_D((i_1, \dots, i_H), (j_1, \dots, j_H)). \end{aligned}$$

Observe that $Q_D((i_1, \dots, i_l), (j_1, \dots, j_l)) = 0$ if $(i_1, \dots, i_l) \neq (j_1, \dots, j_l)$, 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.

Example 1 (ctnd.). We introduce a small example to illustrate the Kronecker representation associated with a CTMC. Consider the following matrices for a three-dimensional problem, respectively with dimensions of 2, 3, and 2 states, and having four terms of Kronecker products:

$$\begin{aligned}
Q_2^{(1)} = Q_3^{(1)} = I_2, \quad Q_1^{(1)} &= \begin{pmatrix} \lambda_1 \\ \mu_1 \end{pmatrix}, \quad Q_4^{(1)} = \begin{pmatrix} \mu \\ \mu \end{pmatrix}, \\
Q_1^{(2)} = Q_3^{(2)} = I_3, \quad Q_2^{(2)} &= \begin{pmatrix} \lambda_2 & & \\ \mu_2 & \lambda_2 & \\ & \mu_2 & \end{pmatrix}, \quad Q_4^{(2)} = \begin{pmatrix} & & \\ & & \\ 1 & & \end{pmatrix}, \\
Q_1^{(3)} = Q_2^{(3)} = I_2, \quad Q_3^{(3)} = Q_4^{(3)} &= \begin{pmatrix} \lambda_3 \\ \mu_3 \end{pmatrix}, \quad Q_4^{(3)} = \begin{pmatrix} & \\ 1 & \end{pmatrix}.
\end{aligned}$$

Then, from (2.1)

$$Q = \sum_{k=1}^4 \bigotimes_{h=1}^3 Q_k^{(h)} + Q_D,$$

and Q is given by (1.1). Now, if we assume that 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 11 for matrices and 12 for diagonal entries, totaling 23, whereas it is 53 for the flat representation in (1.1).

The discrepancy between Kronecker and flat representations becomes substantial for larger values of the state-space size, n , and for denser subsystem matrices. It is also possible to take advantage of identity matrices in the vector–Kronecker product multiplication algorithm discussed in the next section.

Example 1 (ctnd.). Continuing the same example, since $n = 12$ due to $n_1 = n_3 = 2$, and $n_2 = 3$, the recursive definition in (2.2) reveals the nested block structure of Q as follows. At level 0, we have an order 12 matrix [cf. (1.1)]:

$$\begin{array}{cccccccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 1 & 1 & 2 & 2 & 0 & 0 & 1 & 1 & 2 & 2 \\
0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1
\end{array}$$

$$Q = \begin{matrix}
\begin{matrix} 000 \\ 001 \\ 010 \\ 011 \\ 020 \\ 021 \\ 100 \\ 101 \\ 110 \\ 111 \\ 120 \\ 121 \end{matrix} & \begin{pmatrix}
* & \lambda_3 & \lambda_2 & & & \lambda_1 & & & & & & \\
\mu_3 & * & & \lambda_2 & & & \lambda_1 & & & & & \\
\mu_2 & & * & \lambda_3 & \lambda_2 & & & \lambda_1 & & & & \\
& \mu_2 & \mu_3 & * & & \lambda_2 & & & \lambda_1 & & & \\
& & \mu_2 & & * & \lambda_3 & & & & \lambda_1 & & \\
& & & \mu_2 & \mu_3 & * & & & & & \lambda_1 & \\
\mu_1 & & & & & & * & \lambda_3 & \lambda_2 & & & \\
& \mu_1 & & & & & \mu_3 & * & & \lambda_2 & & \\
& & \mu_1 & & & & \mu_2 & & * & \lambda_3 & \lambda_2 & \\
& & & \mu_1 & & & & \mu_2 & \mu_3 & * & & \lambda_2 \\
& & & & \mu_1 & & & & \mu_2 & & * & \lambda_3 \\
\mu & & & & & \mu_1 & & & & \mu_2 & \mu_3 & *
\end{pmatrix}
\end{matrix}
;$$

at level 1, we have a (2×2) block matrix with blocks of order 6:

$$Q = \left(\begin{array}{ccc|ccc} * & \lambda_3 & \lambda_2 & \lambda_1 & & \\ \mu_3 & * & \lambda_2 & \lambda_1 & & \\ \mu_2 & & * & \lambda_3 & \lambda_2 & \\ & \mu_2 & \mu_3 & * & \lambda_2 & \\ & & \mu_2 & & * & \lambda_3 \\ & & \mu_2 & \mu_3 & * & \lambda_1 \\ \hline \mu_1 & & & * & \lambda_3 & \lambda_2 \\ & \mu_1 & & \mu_3 & * & \lambda_2 \\ & & \mu_1 & \mu_2 & * & \lambda_3 & \lambda_2 \\ & & & \mu_2 & \mu_3 & * & \lambda_2 \\ & & & & \mu_2 & * & \lambda_3 \\ \mu & & & & \mu_2 & \mu_3 & * \end{array} \right);$$

at level 2, we have a (6×6) block matrix with blocks of order 2:

$$Q = \left(\begin{array}{ccc|ccc} * & \lambda_3 & \lambda_2 & & \lambda_1 & & \\ \mu_3 & * & \lambda_2 & & \lambda_1 & & \\ \mu_2 & & * & \lambda_3 & \lambda_2 & & \\ & \mu_2 & \mu_3 & * & \lambda_2 & & \\ & & \mu_2 & & * & \lambda_3 & \lambda_1 \\ & & & \mu_2 & \mu_3 & * & \lambda_1 \\ \hline \mu_1 & & & * & \lambda_3 & \lambda_2 & \\ & \mu_1 & & \mu_3 & * & \lambda_2 & \\ & & \mu_1 & & \mu_2 & * & \lambda_3 & \lambda_2 \\ & & & \mu_2 & \mu_3 & * & \lambda_2 \\ & & & & \mu_2 & * & \lambda_3 \\ \mu & & & & \mu_2 & \mu_3 & * \end{array} \right);$$

and finally, at level 3, we have a (12×12) block matrix with blocks of order 1 as in (1.1).

2.2 Vector–Kronecker Product Multiplication Algorithm

The algorithm we discuss next is at the heart of all iterative solvers for sums of Kronecker products. It shows how one can multiply a vector with a Kronecker product of H factors. Here we present its left-oriented version suitable for systems of equations arising from MCs, where the factors are rectangular matrices $X^{(h)} \in \mathbb{R}^{n_h \times m_h}$ for $h = 1, \dots, H$.

The algorithm is based on the identity

$$\bigotimes_{h=1}^H X^{(h)} = \prod_{h=1}^H I_{m_1} \otimes \cdots \otimes I_{m_{h-1}} \otimes X^{(h)} \otimes I_{n_{h+1}} \otimes \cdots \otimes I_{n_H},$$

or, more simply,

$$\bigotimes_{h=1}^H X^{(h)} = \prod_{h=1}^H \left(I_{\prod_{f=1}^{h-1} m_f} \otimes X^{(h)} \otimes I_{\prod_{f=h+1}^H n_f} \right), \quad (2.4)$$

which is due to the *compatibility* of Kronecker product with matrix multiplication [66]. The left multiplication of $\mathbf{x} \in \mathbb{R}^{1 \times \prod_{h=1}^H n_h}$ with $\bigotimes_{h=1}^H X^{(h)}$ yields a product vector whose length ranges from $m_1 \prod_{h=2}^H n_h$ to $\prod_{h=1}^H m_h$ during the course of the multiplication. In (2.4), the h th factor of the form $I_{\prod_{f=1}^{h-1} m_f} \otimes X^{(h)} \otimes I_{\prod_{f=h+1}^H n_f}$ is a rectangular $(\prod_{f=1}^{h-1} m_f \prod_{f=h}^H n_f \times \prod_{f=1}^h m_f \prod_{f=h+1}^H n_f)$ block diagonal matrix having $\prod_{f=1}^{h-1} m_f$ diagonal blocks each of size $(n_h \prod_{f=h+1}^H n_f \times m_h \prod_{f=h+1}^H n_f)$. Furthermore, each of the blocks along the diagonal is an $(n_h \times m_h)$ block matrix, where each subblock is a diagonal matrix of order $\prod_{f=h+1}^H n_f$ with a particular entry of $X^{(h)}$ appearing along its diagonal $\prod_{f=h+1}^H n_f$ many times. It is this feature that is used in devising the algorithm (cf. [66]).

Algorithm 1. *Vector–Kronecker product multiplication algorithm for $\mathbf{x}' = \mathbf{x} \bigotimes_{h=1}^H X^{(h)}$*

Copy $\mathbf{x}$ to $\mathbf{x}'$; $i_{\text{left}} = 1$; $i_{\text{right}} = \prod_{h=2}^H n_h$; $n_{H+1} = 1$;

For $h = 1$ to H ,

$base_i = 0$; $base_j = 0$;

For $i_l = 0, \dots, i_{\text{left}} - 1$,

For $i_r = 0, \dots, i_{\text{right}} - 1$,

$index_i = base_i + i_r$;

For $row = 0, \dots, n_h - 1$,

$\mathbf{z}(row) = \mathbf{x}'(index_i)$; $index_i = index_i + i_{\text{right}}$;

$\mathbf{z}' = \mathbf{zX}^{(h)}$;

$index_j = base_j + i_r$;

For $col = 0, \dots, m_h - 1$,

$\mathbf{x}''(index_j) = \mathbf{z}'(col)$; $index_j = index_j + i_{\text{right}}$;

$base_i = base_i + n_h i_{\text{right}}$; $base_j = base_j + m_h i_{\text{right}}$;

$i_{\text{left}} = i_{\text{left}} m_h$; $i_{\text{right}} = i_{\text{right}} / n_{h+1}$;

Copy $\mathbf{x}''$ to $\mathbf{x}'$.

The only floating-point arithmetic operations (flops) in Algorithm 1 take place in the vector–matrix multiplication $\mathbf{z}' = \mathbf{zX}^{(h)}$, which can be simplified when $X^{(h)} = I_{n_h \times m_h}$. The remaining operations are index manipulation and

copying of vector entries. With regard to space, the algorithm requires two temporary floating-point vectors, $\mathbf{z}$ and $\mathbf{z}'$, that need to be as long as $\max_h(n_h)$ and $\max_h(m_h)$, respectively, and two floating-point vectors, $\mathbf{x}'$ and $\mathbf{x}''$, of length $\max_{h=0,\dots,H}(\prod_{f=1}^h m_f \prod_{f=h+1}^H n_f)$ to compute and return the result. When $X^{(h)} = I_{n_h \times m_h}$, entries of $\mathbf{x}'$ can be directly copied to $\mathbf{x}''$ using appropriate index manipulations. Furthermore, in programming languages that provide handles to memory addresses in the form of pointers, it is possible to carry out the last statement by swapping the two pointers to the arrays $\mathbf{x}'$ and $\mathbf{x}''$ since $\mathbf{x}''$ is to be rewritten in the next turn of the outer loop anyway.

The complexity of a vector multiplication with $\bigotimes_{h=1}^H X^{(h)}$ amounts to

$$2 \sum_{h=1}^H n z_{X^{(h)}} \prod_{f=1}^{h-1} m_f \prod_{f=h+1}^H n_f$$

flops, where $n z_{X^{(h)}}$ is the number of nonzeros in $X^{(h)}$ for $h = 1, \dots, H$. Therefore, the complexity of a vector multiplication with $\sum_{k=1}^K \bigotimes_{h=1}^H X_k^{(h)}$, where $X_k^{(h)} \in \mathbb{R}^{n_h \times m_h}$ for $k = 1, \dots, K$ and $h = 1, \dots, H$, becomes

$$K \prod_{h=1}^H m_h + 2 \sum_{k=1}^K \sum_{h=1}^H n z_{X_k^{(h)}} \prod_{f=1}^{h-1} m_f \prod_{f=h+1}^H n_f.$$

Here, the first term is due to the summation of the K product vectors obtained through Algorithm 1. Note that this expression can be simplified further for Q_O in which the factors are square by substituting n_h for m_h . In that case, one obtains $n(K + 2 \sum_{h=1}^H n z_{Q^{(h)}} / n_h)$, where $n z_{Q^{(h)}} = \sum_k n z_{Q_k^{(h)}}$ and $n z_{Q_k^{(h)}}$ is the number of nonzeros in $Q_k^{(h)}$ for $k = 1, \dots, K$ and $h = 1, \dots, H$.

As was mentioned previously, functional transitions are a feature of SANs and are said to generalize a Kronecker product. Functional transitions enable the elegant modeling of dependencies among subsystems; yet they are not easy to implement efficiently. There is a vector-generalized Kronecker product multiplication algorithm [66] that may be used in the presence of functional transitions. For instance, in the PEPS tool where it is implemented, the different values the functions can take are hard coded into the package. This is done for each problem under consideration. Whenever a function needs to be evaluated in a particular state during the vector-generalized Kronecker product multiplication, the code jumps to a specific location dictated by the problem and carries out a sequence of if statements, trying to find out what the function evaluates to for that state. In practice, this process slows down the code considerably but does not come across when one looks at the time complexity result of vector-generalized Kronecker product multiplication.

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

2.3 Preprocessing

Three techniques can be used to put the Kronecker representation into a more favorable form before solvers take over. These are reordering, grouping, and lumping. We discuss them in order.

We assume that Q represents a time-homogeneous CTMC. Therefore, the left-hand side of each equation in (2.1) is constant up to a symmetric permutation, that is, up to a reordering of the state space, $\mathcal{S}$. 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_O in (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 toward one, the Kronecker representation becomes flatter, implying increased storage requirements. On the other hand, if K were one, then Q could be analyzed along each dimension independently. Hence, K is normally assumed to be larger than one. 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 $Q_k^{(h)}$ matrices denser.

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. The latter corresponds to a symmetric permutation of the factor matrices $Q_k^{(h)}$ for $k = 1, \dots, K$ associated with the renumbered state space $\mathcal{S}^{(h)}$. As indicated in [10,67], 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 have an effect on the convergence of iterative methods sensitive to the nonzero structure [52]. Hence, with 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 factor h contributes to the nonzero structure of Q_O for $h = 1, \dots, H$.

Grouping in MCs based on Kronecker products refers to collapsing the same adjacent 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 investigated in a sequence of papers [10,66,67] on 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 to reduce the state-space size 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 be to group those factors that have functional dependencies among each other. In the absence of functional transitions, it is recommended to group as many factors as possible 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 [32, 34, 54, 85, 146], but a broad, systematic study seems to be lacking.

Lumpability [93] is a property possessed by some MCs that, 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, $\mathcal{S}$, 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 represented by Q is said to be *ordinarily lumpable* with respect to a partitioning of its state space $\mathcal{S} = \bigcup_l \mathcal{S}_l$ and $\mathcal{S}_l \cap \mathcal{S}_u = \emptyset$ for all $l \neq u$ if for all $\mathcal{S}_l \subset \mathcal{S}$ and all $\mathbf{i}, \mathbf{i}' \in \mathcal{S}_l$

$$\sum_{\mathbf{j} \in \mathcal{S}_u} q(\mathbf{i}, \mathbf{j}) = \sum_{\mathbf{j} \in \mathcal{S}_u} q(\mathbf{i}', \mathbf{j}) \text{ for all } \mathcal{S}_u \subset \mathcal{S}. \quad (2.5)$$

A CTMC represented by Q is said to be *exactly lumpable* with respect to a partitioning of its state space $\mathcal{S} = \bigcup_l \mathcal{S}_l$ and $\mathcal{S}_l \cap \mathcal{S}_u = \emptyset$ for all $l \neq u$ if for all $\mathcal{S}_l \subset \mathcal{S}$ and all $\mathbf{i}, \mathbf{i}' \in \mathcal{S}_l$

$$\sum_{\mathbf{j} \in \mathcal{S}_u} q(\mathbf{j}, \mathbf{i}) = \sum_{\mathbf{j} \in \mathcal{S}_u} q(\mathbf{j}, \mathbf{i}') \text{ for all } \mathcal{S}_u \subset \mathcal{S}. \quad (2.6)$$

Ordinary lumpability refers to a partitioning of the state space in which the sums of transition rates from each state in a partition to a(nother) 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 each state of a(nother) partition are the same.

Let $\mathcal{S}_{\text{lumped}}$ denote the lumped state space. On an ordinarily lumped MC one can only compute performance measures defined over $\mathcal{S}_{\text{lumped}}$. On an exactly lumped MC one can compute steady-state performance measures defined over $\mathcal{S}$, transient performance measures defined over $\mathcal{S}_{\text{lumped}}$, and transient performance measures defined over $\mathcal{S}$ 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 (2.6) is harder to satisfy than the ordinary lumpability condition in (2.5). See [20] and references therein for more information regarding the concept of lumpability and its implications.

Lumpability can be investigated on the flat representation of a 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 $O(nz)$ [116]. 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 $\mathcal{S}^{(h)}$ that define the Kronecker representation of Q_O in (2.1) for $h = 1, \dots, H$ independently. For the state space $\mathcal{S}^{(h)}$, detection of ordinary and exact lumpability through partition refinement as in [29] 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 $\mathcal{S}^{(h)}$ and its corresponding matrices $Q_k^{(h)}$ for $k = 1, \dots, K$ with equivalent lumped ones. Lumpability can also be investigated among the state spaces $\mathcal{S}^{(h)}$ that are *replicated* (or identical) with respect to the Kronecker representation of Q_O as in [9]. There, 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 $\mathcal{S}_{\text{lumped}}$ can be computed. Lumpability can also be investigated among the state spaces $\mathcal{S}^{(h)}$ by considering dependencies and matrix properties in the Kronecker representation as in [87, 88]. There, sufficient conditions that satisfy ordinary lumpability are specified and an iterative steady-state solution method that is able to compute performance measures over $\mathcal{S}$ 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 $\mathcal{S}^{(h)}$ totally as in [87, 88] is a very specific kind of performance equivalence and lumping considered in [21, 23], due to its accommodation of functional transitions, it also enables the detection of certain ordinarily lumpable partitionings in which blocks are composed of multiple (nonidentical) state spaces, but the individual state spaces cannot be lumped by themselves. This is not possible with the approaches in [9, 21, 23].

We remark that neither of the two approaches in [9] and [87, 88] that investigate lumpability among the state spaces $\mathcal{S}^{(h)}$ for $h = 1, \dots, H$ is completely automated, uses a Kronecker representation for the lumped MC, and possesses a proper complexity analysis. Furthermore, since the Kronecker representation is rich in structure and the three approaches presented in this chapter do not work on the flat representation, there can very well be other symmetries in the Kronecker representation that also lead to lumpability. Along a slightly different line, the concept of *quasilumpability* [58] can be investigated.

Analyzing Markov Chains using Kronecker Products
Theory and Applications

Dayar, T.

2012, IX, 86 p. 3 illus., Softcover

ISBN: 978-1-4614-4189-2