Chapter 6
Steady-State Analysis

Classical *iterative methods* for the solution of a linear system of equations as in (1.3) start with an initial approximation. At each iteration, they modify the entries of the current approximation in a particular way to obtain a new approximation with the objective that the approximations eventually converge to the true solution [168, 280]. These methods are the building blocks of all advanced iterative methods and can be expressed through the multiplication of the current solution vector at a given iteration with a particular matrix that can be obtained at the outset by splitting the coefficient matrix of the linear system [305, 322], which is Q in our setting. Therefore, we begin by splitting the smaller matrices that form the Kronecker products as in [315] and show how classical iterative methods can be formulated in terms of these smaller matrices. We present block versions of the methods since point versions follow from the block versions by considering blocks of order one.

We continue the discussion with *projection* (or *Krylov subspace*) methods for MCs based on Kronecker products in which approximate solutions satisfying various constraints are extracted from small dimensional subspaces [17, 278, 305]. Being iterative, their basic operation is also vector–Kronecker product multiplication. However, compared to block iterative methods, they require a larger number of supplementary vectors of length equal to the reachable state space size. But, more importantly they need to be used with *preconditioners* to result in effective solvers. Fortunately, the first term of the splitting associated with block iterative methods can be used as preconditioner [60].

In [43, 49, 50, 52], *aggregation–disaggregation* steps are coupled with various iterative methods for MCs based on Kronecker products to accelerate convergence. An *iterative aggregation–disaggregation* (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 [47] and [48], respectively. The adaptive IAD method in [48] is

T. Dayar, *Kronecker Modeling and Analysis of Multidimensional Markovian Systems*, Springer Series in Operations Research and Financial Engineering, https://doi.org/10.1007/978-3-319-97129-2_6

improved in [53] through a recursive definition and called *multilevel* (ML). Here, we present this simple ML method and then discuss a class of ML methods based on it which are shown to be quite effective [59, 61] in solving a large number of problems in the literature. ML methods can easily use iterative methods based on splittings at each level before aggregation and after disaggregation.

Matrix analytic methods are geared toward MCs having state spaces that can be partitioned into subsets called *levels*. For such MCs, the transition matrix when symmetrically permuted according to increasing level number should also have a particular nonzero structure, such as *block tridiagonal* or *block Hessenberg*. For instance, the well-known *quasi-birth-and-death processes* (QBDs) fall under the class of processes which lend themselves to steady-state analysis with matrix analytic methods. In this context, matrix analytic methods are those for which equations involving matrices are set up and solutions are expressed in terms of matrices [166]. These methods were originally proposed [251, 252] for processes with PH distributions and then improved over the years [32, 166, 214]. They characterize the solution by matrices having stochastic interpretations and sizes determined by the number of states within levels. Here, we consider CTMCs and concentrate on the class of *level-dependent* QBDs (LDQBDs). We show how the systems of stochastic chemical kinetics modeled using Kronecker products can be expressed as infinite LDQBDs, and analyzed for their steady-state with the help of Lyapunov functions. In passing, we remark that the concept of level introduced here has nothing to do with the level concept introduced during the discussion of iterative solution methods.

Decompositonal iterative methods compute steady-state solutions by decomposing a model into its submodels, analyzing the submodels individually for their steady-state, and putting back the individual solutions together in an iterative computational framework. As such, they may aim at obtaining the steady-state solution exactly up to computer precision, approximatively when a few digits of accuracy is sufficient, or within upper and lower bounds. Methods of the first two kinds are discussed on the simple availability model in Example 1 [15] and a class of closed queueing networks [104].

Finally, we describe how compact solution vectors in HTD format can be used in the analysis of point iterative methods and projection methods following the recent results in [64, 66].

6.1 Block Iterative Methods

We begin by splitting submodel transition matrices into three terms as in

$$Q_k^{(h)} = D_k^{(h)} + U_k^{(h)} + L_k^{(h)} \quad \text{for } k \in \mathcal{K} \text{ and } h = 1, \ldots, H \,, \tag{6.1}$$

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 $D_k^{(h)} \geqslant 0$, $U_k^{(h)} \geqslant 0$, and $L_k^{(h)} \geqslant 0$ since $Q_k^{(h)} \geqslant 0$.

Now, let us denote the off-diagonal part of Q in (2.10) as

$$Q_O = Q - Q_D \quad \text{so that} \quad Q_O(p,w) = \sum_{k \in \mathcal{K}_{p,w}} Q_k(p,w), \tag{6.2}$$

where

$$Q_k(p,w) = \alpha_k \bigotimes_{h=1}^{H} Q_k^{(h)}(\mathcal{R}_p^{(h)}, \mathcal{R}_p^{(h)}) \quad \text{for } p,w = 0,\ldots,N-1\,.$$

Observe that $Q_O(p,w)$ for $p > w$ is in the strictly block lower-triangular part of Q_O, while $Q_O(p,w)$ for $p < w$ is in the strictly block upper-triangular part of Q_O. Hence, all that needs to be done is to provide block splittings of $Q_O(p,p)$ for $p = 0,\ldots,N-1$.

Using Lemma A.8 in [315], which rests on the associativity of Kronecker product and the *distributivity* of Kronecker product over matrix addition, it is possible to express diagonal block $Q_O(p,p)$ in Q_O at level $l = 0,\ldots,H$ using (6.1) as

$$Q_O(p,p) = Q(p,p)_{U(l)} + Q(p,p)_{L(l)} + Q(p,p)_{DU(l)} + Q(p,p)_{DL(l)}\,, \tag{6.3}$$

where

$$Q(p,p)_{U(l)} = \sum_{k \in \mathcal{K}_{p,p}} \sum_{h=1}^{l} \alpha_k \left(\bigotimes_{h'=1}^{h-1} D_k^{(h')}(\mathcal{R}_p^{(h')}, \mathcal{R}_p^{(h')}) \right) \otimes U_k^{(h)}(\mathcal{R}_p^{(h)}, \mathcal{R}_p^{(h)}) \otimes \left(\bigotimes_{h'=h+1}^{H} Q_k^{(h')}(\mathcal{R}_p^{(h')}, \mathcal{R}_p^{(h')}) \right)\,, \tag{6.4}$$

$$Q(p,p)_{L(l)} = \sum_{k \in \mathcal{K}_{p,p}} \sum_{h=1}^{l} \alpha_k \left(\bigotimes_{h'=1}^{h-1} D_k^{(h')}(\mathcal{R}_p^{(h')}, \mathcal{R}_p^{(h')}) \right) \otimes L_k^{(h)}(\mathcal{R}_p^{(h)}, \mathcal{R}_p^{(h)}) \otimes \left(\bigotimes_{h'=h+1}^{H} Q_k^{(h')}(\mathcal{R}_p^{(h')}, \mathcal{R}_p^{(h')}) \right) \tag{6.5}$$

correspond, respectively, to the strictly block upper- and block lower-triangular parts of $Q_O(p,p)$ at level l, and

$$Q(p,p)_{DU(l)} = \sum_{k \in \mathcal{K}_{p,p}} \sum_{h=l+1}^{H} \alpha_k \left(\bigotimes_{h'=1}^{h-1} D_k^{(h')}(\mathcal{R}_p^{(h')}, \mathcal{R}_p^{(h')}) \right) \otimes U_k^{(h)}(\mathcal{R}_p^{(h)}, \mathcal{R}_p^{(h)}) \otimes \left(\bigotimes_{h'=h+1}^{H} Q_k^{(h')}(\mathcal{R}_p^{(h')}, \mathcal{R}_p^{(h')}) \right)\,, \tag{6.6}$$

$$Q(p,p)_{DL(l)} = \sum_{k\in\mathcal{K}_{p,p}} \sum_{h=l+1}^{H} \alpha_k \left(\bigotimes_{h'=1}^{h-1} D_k^{(h')}(\mathcal{R}_p^{(h')}, \mathcal{R}_p^{(h')}) \right) \otimes L_k^{(h)}(\mathcal{R}_p^{(h)}, \mathcal{R}_p^{(h)}) \otimes \left(\bigotimes_{h'=h+1}^{H} Q_k^{(h')}(\mathcal{R}_p^{(h')}, \mathcal{R}_p^{(h')}) \right) . \tag{6.7}$$

correspond, respectively, to the strictly upper- and strictly lower-triangular parts of the *block diagonal* of $Q_O(p,p)$ at level l. Observe that $Q(p,p)_{U(l)} \geqslant 0$, $Q(p,p)_{L(l)} \geqslant 0$, $Q(p,p)_{DU(l)} \geqslant 0$, and $Q(p,p)_{DL(l)} \geqslant 0$. Furthermore, we remark that $l = 0$ implies $Q_O(p,p)$ is a single block for which $Q(p,p)_{U(0)} = Q(p,p)_{L(0)} = 0$, whereas $l = H$ corresponds to a point-wise partitioning of $Q_O(p,p)$ for which $Q(p,p)_{DU(H)} = Q(p,p)_{DL(H)} = 0$. Hence, for iterative methods based on *block partitionings* $l = 1, \ldots, H-1$ should be used. In passing to an example, we remark that in a non-Kronecker setting, one could obtain block partitionings of Q as discussed in [78, 123, 253]. Here, it is the Kronecker structure of Q that suggests suitable block partitionings.

Example 2. (ctnd.) Consider the block partitioning of our problem at level 0 for which $l = 0$ and Q_O is viewed as a (4×4) block matrix with four blocks, respectively, of order 4, 2, 4, and 2 along the diagonal. Then for $p = 0, 1, 2, 3$ from (6.4) and (6.5), we have

$$Q(p,p)_{U(0)} = 0 \,, \quad Q(p,p)_{L(0)} = 0 \,, \quad Q(p,p)_{U(0)} + Q(p,p)_{L(0)} = 0 \,,$$

whereas from (6.6) and (6.7), we have

$$\begin{aligned} Q(p,p)_{DU(0)} = \sum_{k\in\mathcal{K}_{p,p}} \alpha_k \Big(& U_k^{(1)}(\mathcal{R}_p^{(1)}, \mathcal{R}_p^{(1)}) \otimes Q_k^{(2)}(\mathcal{R}_p^{(2)}, \mathcal{R}_p^{(2)}) \otimes Q_k^{(3)}(\mathcal{R}_p^{(3)}, \mathcal{R}_p^{(3)}) \\ & + D_k^{(1)}(\mathcal{R}_p^{(1)}, \mathcal{R}_p^{(1)}) \otimes U_k^{(2)}(\mathcal{R}_p^{(2)}, \mathcal{R}_p^{(2)}) \otimes Q_k^{(3)}(\mathcal{R}_p^{(3)}, \mathcal{R}_p^{(3)}) \\ & + D_k^{(1)}(\mathcal{R}_p^{(1)}, \mathcal{R}_p^{(1)}) \otimes D_k^{(2)}(\mathcal{R}_p^{(2)}, \mathcal{R}_p^{(2)}) \otimes U_k^{(3)}(\mathcal{R}_p^{(3)}, \mathcal{R}_p^{(3)}) \Big) \,, \\ Q(p,p)_{DL(0)} = \sum_{k\in\mathcal{K}_{p,p}} \alpha_k \Big(& L_k^{(1)}(\mathcal{R}_p^{(1)}, \mathcal{R}_p^{(1)}) \otimes Q_k^{(2)}(\mathcal{R}_p^{(2)}, \mathcal{R}_p^{(2)}) \otimes Q_k^{(3)}(\mathcal{R}_p^{(3)}, \mathcal{R}_p^{(3)}) \\ & + D_k^{(1)}(\mathcal{R}_p^{(1)}, \mathcal{R}_p^{(1)}) \otimes L_k^{(2)}(\mathcal{R}_p^{(2)}, \mathcal{R}_p^{(2)}) \otimes Q_k^{(3)}(\mathcal{R}_p^{(3)}, \mathcal{R}_p^{(3)}) \\ & + D_k^{(1)}(\mathcal{R}_p^{(1)}, \mathcal{R}_p^{(1)}) \otimes D_k^{(2)}(\mathcal{R}_p^{(2)}, \mathcal{R}_p^{(2)}) \otimes L_k^{(3)}(\mathcal{R}_p^{(3)}, \mathcal{R}_p^{(3)}) \Big) \,, \end{aligned}$$

$$Q(p,p)_{DU(0)} + Q(p,p)_{DL(0)} = Q_O(p,p) \,.$$

Now consider the block partitioning of the problem at level 1 for which $l = 1$ and $Q_O(0,0)$ is viewed as a (2×2) block matrix with blocks of order 2, $Q_O(1,1)$ is viewed as a (1×1) block matrix with a block of order 2, $Q_O(2,2)$ is viewed as a (2×2) block matrix with blocks of order 2, and $Q_O(3,3)$ is viewed as a (2×2) block matrix with blocks of order 1 (see Table 2.1). Then from (6.4) and (6.5), we have

$$Q(p,p)_{U(1)} = \sum_{k\in\mathcal{K}_{p,p}} \alpha_k U_k^{(1)}(\mathcal{R}_p^{(1)},\mathcal{R}_p^{(1)}) \otimes Q_k^{(2)}(\mathcal{R}_p^{(2)},\mathcal{R}_p^{(2)}) \otimes Q_k^{(3)}(\mathcal{R}_p^{(3)},\mathcal{R}_p^{(3)}) \,,$$
$$Q(p,p)_{L(1)} = \sum_{k\in\mathcal{K}_{p,p}} \alpha_k L_k^{(1)}(\mathcal{R}_p^{(1)},\mathcal{R}_p^{(1)}) \otimes Q_k^{(2)}(\mathcal{R}_p^{(2)},\mathcal{R}_p^{(2)}) \otimes Q_k^{(3)}(\mathcal{R}_p^{(3)},\mathcal{R}_p^{(3)}) \,.$$

Consequently,

$$Q(0,0)_{U(1)} + Q(0,0)_{L(1)} = \begin{pmatrix} & \lambda_1 \\ & \lambda_1 \\ \mu_1 & \\ \mu_1 & \end{pmatrix}, \quad Q(1,1)_{U(1)} + Q(1,1)_{L(1)} = 0,$$

$$Q(2,2)_{U(1)} + Q(2,2)_{L(1)} = \begin{pmatrix} & \\ & \\ \mu_1 & \\ \mu_1 & \end{pmatrix}, \quad \text{and}$$
$$Q(3,3)_{U(1)} + Q(3,3)_{L(1)} = \begin{pmatrix} & \\ \mu_1 & \end{pmatrix} \,,$$

whereas from (6.6) and (6.7), we have

$$\begin{aligned} Q(p,p)_{DU(1)} = \sum_{k\in\mathcal{K}_{p,p}} \alpha_k \Big(& D_k^{(1)}(\mathcal{R}_p^{(1)},\mathcal{R}_p^{(1)}) \otimes U_k^{(2)}(\mathcal{R}_p^{(2)},\mathcal{R}_p^{(2)}) \otimes Q_k^{(3)}(\mathcal{R}_p^{(3)},\mathcal{R}_p^{(3)}) \\ & + D_k^{(1)}(\mathcal{R}_p^{(1)},\mathcal{R}_p^{(1)}) \otimes D_k^{(2)}(\mathcal{R}_p^{(2)},\mathcal{R}_p^{(2)}) \otimes U_k^{(3)}(\mathcal{R}_p^{(3)},\mathcal{R}_p^{(3)}) \Big) \,, \\ Q(p,p)_{DL(1)} = \sum_{k\in\mathcal{K}_{p,p}} \alpha_k \Big(& D_k^{(1)}(\mathcal{R}_p^{(1)},\mathcal{R}_p^{(1)}) \otimes L_k^{(2)}(\mathcal{R}_p^{(2)},\mathcal{R}_p^{(2)}) \otimes Q_k^{(3)}(\mathcal{R}_p^{(3)},\mathcal{R}_p^{(3)}) \\ & + D_k^{(1)}(\mathcal{R}_p^{(1)},\mathcal{R}_p^{(1)}) \otimes D_k^{(2)}(\mathcal{R}_p^{(2)},\mathcal{R}_p^{(2)}) \otimes L_k^{(3)}(\mathcal{R}_p^{(3)},\mathcal{R}_p^{(3)}) \Big) \,. \end{aligned}$$

Consequently,

$$Q(0,0)_{DU(1)} + Q(0,0)_{DL(1)} = \begin{pmatrix} & 2\lambda_3 & \\ \mu_3 & & \\ & & 2\lambda_3 \\ & \mu_3 & \end{pmatrix},$$
$$Q(1,1)_{DU(1)} + Q(1,1)_{DL(1)} = \begin{pmatrix} & \\ \mu_3 & \end{pmatrix} \,,$$
$$Q(2,2)_{DU(1)} + Q(2,2)_{DL(1)} = \begin{pmatrix} \mu_3 & & \\ & & \\ & \mu_3 & \end{pmatrix}, \quad \text{and}$$
$$Q(3,3)_{DU(1)} + Q(3,3)_{DL(1)} = 0 \,.$$

Finally, consider the block partitioning of the same example at level 2 for which $l = 2$. Then from (6.4) and (6.5), we have

$$\begin{aligned}
Q(p,p)_{U(2)} = \sum_{k\in\mathcal{K}_{p,p}} \alpha_k \Big(& U_k^{(1)}(\mathcal{R}_p^{(1)},\mathcal{R}_p^{(1)}) \otimes Q_k^{(2)}(\mathcal{R}_p^{(2)},\mathcal{R}_p^{(2)}) \otimes Q_k^{(3)}(\mathcal{R}_p^{(3)},\mathcal{R}_p^{(3)}) \\
& + D_k^{(1)}(\mathcal{R}_p^{(1)},\mathcal{R}_p^{(1)}) \otimes U_k^{(2)}(\mathcal{R}_p^{(2)},\mathcal{R}_p^{(2)}) \otimes Q_k^{(3)}(\mathcal{R}_p^{(3)},\mathcal{R}_p^{(3)}) \Big), \\
Q(p,p)_{L(2)} = \sum_{k\in\mathcal{K}_{p,p}} \alpha_k \Big(& L_k^{(1)}(\mathcal{R}_p^{(1)},\mathcal{R}_p^{(1)}) \otimes Q_k^{(2)}(\mathcal{R}_p^{(2)},\mathcal{R}_p^{(2)}) \otimes Q_k^{(3)}(\mathcal{R}_p^{(3)},\mathcal{R}_p^{(3)}) \\
& + D_k^{(1)}(\mathcal{R}_p^{(1)},\mathcal{R}_p^{(1)}) \otimes L_k^{(2)}(\mathcal{R}_p^{(2)},\mathcal{R}_p^{(2)}) \otimes Q_k^{(3)}(\mathcal{R}_p^{(3)},\mathcal{R}_p^{(3)}) \Big),
\end{aligned}$$

whereas from (6.6) and (6.7), we have

$$\begin{aligned}
Q(p,p)_{DU(2)} &= \sum_{k\in\mathcal{K}_{p,p}} \alpha_k D_k^{(1)}(\mathcal{R}_p^{(1)},\mathcal{R}_p^{(1)}) \otimes D_k^{(2)}(\mathcal{R}_p^{(2)},\mathcal{R}_p^{(2)}) \otimes U_k^{(3)}(\mathcal{R}_p^{(3)},\mathcal{R}_p^{(3)}), \\
Q(p,p)_{DL(2)} &= \sum_{k\in\mathcal{K}_{p,p}} \alpha_k D_k^{(1)}(\mathcal{R}_p^{(1)},\mathcal{R}_p^{(1)}) \otimes D_k^{(2)}(\mathcal{R}_p^{(2)},\mathcal{R}_p^{(2)}) \otimes L_k^{(3)}(\mathcal{R}_p^{(3)},\mathcal{R}_p^{(3)}).
\end{aligned}$$

Note that at this level, as in $l = 1$, for this particular example $Q_O(0,0)$ is viewed as a (2×2) block matrix with blocks of order 2, $Q_O(1,1)$ is viewed as a (1×1) block matrix with a block of order 2, $Q_O(2,2)$ is viewed as a (2×2) block matrix with blocks of order 2, and $Q_O(3,3)$ is viewed as a (2×2) block matrix with blocks of order 1 (see Table 2.1). Hence,

$$\begin{aligned}
Q(0,0)_{U(2)} + Q(0,0)_{L(2)} &= Q(0,0)_{U(1)} + Q(0,0)_{L(1)}\ , \\
Q(1,1)_{U(2)} + Q(1,1)_{L(2)} &= Q(1,1)_{U(1)} + Q(1,1)_{L(1)} = 0, \\
Q(2,2)_{U(2)} + Q(2,2)_{L(2)} &= Q(2,2)_{U(1)} + Q(2,2)_{L(1)}\ , \\
Q(3,3)_{U(2)} + Q(3,3)_{L(2)} &= Q(3,3)_{U(1)} + Q(3,3)_{L(1)}\ , \\
Q(0,0)_{DU(2)} + Q(0,0)_{DL(2)} &= Q(0,0)_{DU(1)} + Q(0,0)_{DL(1)}\ , \\
Q(1,1)_{DU(2)} + Q(1,1)_{DL(2)} &= Q(1,1)_{DU(1)} + Q(1,1)_{DL(1)}\ , \\
Q(2,2)_{DU(2)} + Q(2,2)_{DL(2)} &= Q(2,2)_{DU(1)} + Q(2,2)_{DL(1)}\ , \\
Q(3,3)_{DU(2)} + Q(3,3)_{DL(2)} &= Q(3,3)_{DU(1)} + Q(3,3)_{DL(1)} = 0.
\end{aligned}$$

At $l = 3$, diagonal blocks $Q_O(0,0)$, $Q_O(1,1)$, $Q_O(2,2)$, and $Q_O(3,3)$ of Q_O are, respectively, (4×4), (2×2), (4×4), and (2×2) block matrices with blocks of order 1.

Now, let Q in (2.10) be irreducible and *split* at level l using (6.3) as

$$Q = Q_O + Q_D = Q_{U(l)} + Q_{L(l)} + Q_{DU(l)} + Q_{DL(l)} + Q_D = M - W\ , \quad (6.8)$$

where

$$Q_{U(l)} = \begin{pmatrix} Q(0,0)_{U(l)} & Q(0,1) & \cdots & Q(0,N-1) \\ & Q(1,1)_{U(l)} & \cdots & Q(1,N-1) \\ & & \ddots & \vdots \\ & & & Q(N-1,N-1)_{U(l)} \end{pmatrix}$$

is the strictly block upper-triangular part of Q_O at level l,

$$Q_{L(l)} = \begin{pmatrix} Q(0,0)_{L(l)} & & & \\ \vdots & \ddots & & \\ Q(N-2,0) & \cdots & Q(N-2,N-2)_{L(l)} & \\ Q(N-1,0) & \cdots & Q(N-1,N-2) & Q(N-1,N-1)_{L(l)} \end{pmatrix}$$

is the strictly block lower-triangular part of Q_O at level l,

$$Q_{DU(l)} = \begin{pmatrix} Q(0,0)_{DU(l)} & & \\ & \ddots & \\ & & Q(N-1,N-1)_{DU(l)} \end{pmatrix}$$

is the strictly upper-triangular part of the block diagonal of Q_O at level l,

$$Q_{DL(l)} = \begin{pmatrix} Q(0,0)_{DL(l)} & & \\ & \ddots & \\ & & Q(N-1,N-1)_{DL(l)} \end{pmatrix}$$

is the strictly lower-triangular part of the block diagonal of Q_O at level l, and M is nonsingular (i.e., M^{-1} exists).

Then *power*, *block Jacobi over-relaxation* (BJOR), and *block successive over-relaxation* (BSOR) methods are based on different splittings of Q [322], and each method is in the form

$$\boldsymbol{\pi}_{(it+1)} := \boldsymbol{\pi}_{(it)} T \quad \text{for} \quad it = 0, \ldots, \texttt{maxit} - 1$$

with the sequence of approximations $\boldsymbol{\pi}_{(it+1)}$ to the steady-state vector $\boldsymbol{\pi}$ in (1.3), where $\boldsymbol{\pi}_{(0)} \in \mathbb{R}_{>0}^{1\times|\mathcal{R}|}$ is the initial approximation such that $\boldsymbol{\pi}_{(0)}\mathbf{e} = 1$ and

$$T = WM^{-1}$$

is the *iteration matrix*.

Note that T does not change from iteration to iteration, and only the current approximation $\boldsymbol{\pi}_{(it)}$ is used to compute the new approximation $\boldsymbol{\pi}_{(it+1)}$. Hence, these methods 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 [158, 242] of T in magnitude is 1, that is, the *spectral radius* of T, $\rho(T)$, is equal to 1. In order to ensure convergence, T should not have other eigenvalues with magnitude 1, that is,

T should be *aperiodic*. For a converging iteration, the magnitude of the eigenvalue of T closest to 1, that is, its *subdominant eigenvalue*, determines the *rate of convergence* [17, 30, 229, 305, 322]. When the subdominant eigenvalue of T is close to 1 in magnitude, slow convergence is witnessed.

Given a DTMC with one-step transition probability matrix P such that $P \in \mathbb{R}_{\geqslant 0}^{|\mathcal{R}|\times|\mathcal{R}|}$ and $P\mathbf{e} = \mathbf{e}$ [194, 305], one can conceive of block iterative methods for $Q = P - I$ as those defined by (6.8) to compute the steady-state vector $\boldsymbol{\pi}$ which satisfies

$$\boldsymbol{\pi}P = \boldsymbol{\pi}, \quad \boldsymbol{\pi}\mathbf{e} = 1. \tag{6.9}$$

In practice, an explicit inversion of M does not take place. Instead, at iteration it, one solves the consistent linear system

$$\boldsymbol{\pi}_{(it+1)}M = \boldsymbol{\pi}_{(it)}W$$

with coefficient matrix M for the unknown vector $\boldsymbol{\pi}_{(it+1)}$ using the right-hand side vector $\boldsymbol{\pi}_{(it)}W$. The iteration stops if the norm of the error vector (or alternatively, the norm of the residual vector) (see (5.4)) is less than a prespecified tolerance, `stop_tol`, a run time limit, `time_limit`, is reached, or a maximum number of iterations, `maxit`, is performed. Otherwise, the iteration number it is incremented and the iteration continues.

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

$$\begin{aligned}
M^{Power} &= -\Gamma I\,, \\
W^{Power} &= -\Gamma\left(I + \frac{1}{\Gamma}Q\right)\,, \\
M^{BJOR} &= \frac{1}{\omega}(Q_D + Q_{DU(l)} + Q_{DL(l)})\,, \\
W^{BJOR} &= \frac{1-\omega}{\omega}(Q_D + Q_{DU(l)} + Q_{DL(l)}) - Q_{U(l)} - Q_{L(l)}\,, \\
M^{BSOR} &= \frac{1}{\omega}(Q_D + Q_{DU(l)} + Q_{DL(l)}) + Q_{U(l)}\,, \\
W^{BSOR} &= \frac{1-\omega}{\omega}(Q_D + Q_{DU(l)} + Q_{DL(l)}) - Q_{L(l)}\,,
\end{aligned}$$

where $\Gamma \in [\max_{i\in\mathcal{R}} |q_D(i,i)|, \infty)$ is the *uniformization parameter* of power method and $\omega \in (0,2)$ is the *relaxation parameter* of BJOR and BSOR methods. Here, forward iteration refers to computing unknowns ordered toward the beginning of the reachable state space earlier than unknowns ordered later in the reachable state space. Power method works at level $l = H$ since it is a point method. Furthermore, BJOR and BSOR reduce to *block Jacobi* (BJacobi) and *block Gauss–Seidel* (BGS) methods for $\omega = 1$, and they become point JOR and point SOR methods for $l = H$. We remark that [173] shows how one can find $\max_{i\in\mathcal{R}} |q_D(i,i)|$ in the presence of functional transitions when Q_D is given as a sum of Kronecker products. It is possible to use the same approach in each $Q_D(p,p)$ for $p = 0, \ldots, N-1$ when there are multiple reachable state space partitions.

When Q is irreducible and $\omega \in (0,1)$, for JOR and SOR we have $M^{-1}, W \in \mathbb{R}_{\geqslant 0}^{|\mathcal{R}|\times|\mathcal{R}|}$, $\rho(T) = 1$, T is irreducible and aperiodic. Hence, JOR and SOR can be made to converge by choosing $\omega \in (0,1)$. To avoid periodicity of T^{GS} for the point GS method, one can symmetrically permute Q such that it is in block lower-Hessenberg form (i.e., $Q(p,w) = 0$ if $p+1 < w$) and all subdiagonal blocks have at least one nonzero in each row with the last diagonal block being (1×1) [96]. When Q is irreducible, $T^{BJacobi}$ and T^{BGS} satisfy $M^{-1}, W \in \mathbb{R}_{\geqslant 0}^{|\mathcal{R}|\times|\mathcal{R}|}$ and $\rho(T) = 1$ is a simple eigenvalue [16, 230, 232, 233, 249].

Now, let us assume that $T \in \mathbb{R}_{\geqslant 0}^{|\mathcal{R}|\times|\mathcal{R}|}$ is irreducible, but periodic having $N' \in \mathbb{Z}_{>0}$ *periodic classes*, without loss of generality, as in

$$T = \begin{pmatrix} & T(0,1) & & \\ & & \ddots & \\ & & & T(N'-2,N'-1) \\ T(N'-1,0) & & & \end{pmatrix}.$$

Note that $N' = 1$ is the aperiodic case. BJacobi and BGS may yield a periodic T with $N' > 1$ when

$$Q = \begin{pmatrix} Q(0,0) & Q(0,1) & & \\ & \ddots & \ddots & \\ & & Q(N'-2,N'-2) & Q(N'-2,N'-1) \\ Q(N'-1,0) & & & Q(N'-1,N'-1) \end{pmatrix}.$$

An often overlooked result in this context is that when none of the diagonal blocks of Q can be symmetrically permuted to block diagonal form, $T^{BJacobi}$ and T^{BGS} will have states of each partition in the same periodic class [87]. Researchers have looked into ways of avoiding periodicity of T and accelerating convergence. This is something to which we return in the next sections.

Since $Q = Q_O + Q_D$, power method at iteration it can be expressed as

$$\boldsymbol{\pi}_{(it+1)} := \boldsymbol{\pi}_{(it)} + \frac{1}{\Gamma}\left(\boldsymbol{\pi}_{(it)}Q_D + \boldsymbol{\pi}_{(it)}Q_O\right) . \tag{6.10}$$

Observe that the second term in (6.10) 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 expressed using sums of Kronecker products (see (6.2)).

The BJOR method with a level l block partitioning at iteration it satisfies

$$\begin{aligned} &\boldsymbol{\pi}_{(it+1)}(Q_D + Q_{DU(l)} + Q_{DL(l)}) \\ &\quad = (1-\omega)\left(\boldsymbol{\pi}_{(it)}Q_D + \boldsymbol{\pi}_{(it)}Q_{DU(l)} + \boldsymbol{\pi}_{(it)}Q_{DL(l)}\right) \\ &\qquad -\omega\left(\boldsymbol{\pi}_{(it)}Q_{U(l)} + \boldsymbol{\pi}_{(it)}Q_{L(l)}\right) . \end{aligned} \tag{6.11}$$

This is a block diagonal linear system with 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 expressed using sums of Kronecker products (see (6.4), (6.5), (6.6), and (6.7)). In particular, there are $\prod_{h=1}^{l} |\mathcal{R}_p^{(h)}|$ blocks of order $\prod_{h=l+1}^{H} |\mathcal{R}_p^{(h)}|$ along the diagonal of $Q_D(p,p) + Q(p,p)_{DU(l)} + Q(p,p)_{DL(l)}$ for $p = 0, \ldots, N-1$. Hence, (6.11) is equivalent to $\sum_{p=0}^{N-1} \prod_{h=1}^{l} |\mathcal{R}_p^{(h)}|$ independent, nonsingular linear systems with nonzero right-hand sides where the linear systems with coefficient matrices in $Q_D(p,p) + Q(p,p)_{DU(l)} + Q(p,p)_{DL(l)}$ are each of order $\prod_{h=l+1}^{H} |\mathcal{R}_p^{(h)}|$. If there is space, one can generate and factorize in sparse storage the nonsingular blocks of the form

$$Q((i_1,\ldots,i_l),(i_1,\ldots,i_l)) = \sum_{k \in \mathcal{K}_{p,p}} \alpha_k \left(\prod_{h=1}^{l} q_k^{(h)}(i_h,i_h) \right) \left(\bigotimes_{h=l+1}^{H} Q_k^{(h)}(\mathcal{R}_p^{(h)},\mathcal{R}_p^{(h)}) \right)$$

$$+ \; Q_D((i_1,\ldots,i_l),(i_1,\ldots,i_l)) \quad \text{for } (i_1,\ldots,i_l) \in \bigtimes_{h=1}^{l} \mathcal{R}_p^{(h)} \text{ and } p = 0,\ldots,N-1 \tag{6.12}$$

along the diagonal (see (2.4)) of $(Q_D + Q_{DU(l)} + Q_{DL(l)})$ at the outset and solve the $\sum_{p=0}^{N-1} |\bigtimes_{h=1}^{l} \mathcal{R}_p^{(h)}|$ systems directly at each iteration. Otherwise, one can use an iterative method, even a block iterative method, such as BJOR, since the off-diagonal parts of diagonal blocks given by

$$\sum_{k \in \mathcal{K}_{p,p}} \alpha_k \left(\prod_{h=1}^{l} q_k^{(h)}(i_h,i_h) \right) \left(\bigotimes_{h=l+1}^{H} Q_k^{(h)}(\mathcal{R}_p^{(h)},\mathcal{R}_p^{(h)}) \right)$$

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 it, we have

$$\boldsymbol{\pi}_{(it+1)}(Q_D + Q_{DU(l)} + Q_{DL(l)} + \omega Q_{U(l)}) \tag{6.13}$$

$$= (1-\omega)\left(\boldsymbol{\pi}_{(it)} Q_D + \boldsymbol{\pi}_{(it)} Q_{DU(l)} + \boldsymbol{\pi}_{(it)} Q_{DL(l)}\right) - \omega \boldsymbol{\pi}_{(it)} Q_{L(l)} \; .$$

This is a block upper-triangular linear system with 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 $Q_{L(l)}$, $Q_{DU(l)}$, and $Q_{DL(l)}$ are expressed using sums of Kronecker products. In particular, there are $\prod_{h=1}^{l} |\mathcal{R}_p^{(h)}|$ blocks of order $\prod_{h=l+1}^{H} |\mathcal{R}_p^{(h)}|$ along the diagonal of $Q_D(p,p) + Q(p,p)_{DU(l)} + Q(p,p)_{DL(l)} + \omega Q(p,p)_{U(l)}$ for $p = 0, \ldots, N-1$. In [315], 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* SOR. Therein, a version of the same algorithm for backward BSOR is also discussed. Here we remark that a non-recursive block upper-triangular solution algorithm for (6.13) is also possible [58] and a block row-oriented version is preferable in the presence of functional transitions as in Algorithm 6.

Observe in Algorithm 6 that initially the nonzero right-hand side subvector $\mathbf{b}(\mathcal{R}_p)$ can be efficiently computed using the vector–Kronecker product multiplication algorithm, since $Q_{L(l)}$, $Q_{U(l)}$, $Q_{DL(l)}$, and $Q_{DU(l)}$ are expressed using sums of Kronecker products. Furthermore, $Q((i_1,\ldots,i_l),(i_1,\ldots,i_l))$ for $(i_1,\ldots,i_l) \in \times_{h=1}^{l} \mathcal{R}_p^{(h)}$ and $p = 0,\ldots,N-1$ is given in (6.12) in terms of a sum of Kronecker products, and $Q((i_1,\ldots,i_l),(j_1,\ldots,j_l))$ for $(j_1,\ldots,j_l) \in \times_{h=1}^{l} \mathcal{R}_p^{(h)}$ and $(j_1,\ldots,j_l) > (i_1,\ldots,i_l)$ can be expressed in terms of a sum of Kronecker products using (6.4) as

$$
\begin{aligned}
&Q((i_1,\ldots,i_l),(j_1,\ldots,j_l)) \\
&= \sum_{k\in K_{p,p}} \alpha_k \sum_{h=1}^{l} \left(\prod_{h'=1}^{h-1} d_k^{(h')}(i_{h'},i_{h'}) \right) u_k^{(h)}(i_h,j_h) \left(\prod_{h'=h+1}^{l} q_k^{(h')}(i_{h'},j_{h'}) \right) \\
&\qquad \left(\bigotimes_{h'=l+1}^{H} Q_k^{(h')}(\mathcal{R}_p^{(h)},\mathcal{R}_p^{(h)}) \right) .
\end{aligned}
$$

Algorithm 6. *Non-recursive block upper-triangular solution at level l in iteration it of BSOR for MCs based on Kronecker products.*

For reachable state space partition $p := 0,\ldots,N-1$,
$\mathbf{b}(\mathcal{R}_p) := (1-\omega)\left(\boldsymbol{\pi}_{(it)}(\mathcal{R}_p)Q_D(p,p) + \boldsymbol{\pi}_{(it)}(\mathcal{R}_p)Q(p,p)_{DU(l)} + \boldsymbol{\pi}_{(it)}(\mathcal{R}_p)Q(p,p)_{DL(l)}\right)$
$-\omega\left(\boldsymbol{\pi}_{(it)}(\mathcal{R}_p)Q(p,p)_{L(l)} + \sum_{w=p+1}^{N-1} \boldsymbol{\pi}_{(it)}(\mathcal{R}_w)Q(w,p) + \sum_{w=0}^{p-1} \boldsymbol{\pi}_{(it+1)}(\mathcal{R}_w)Q(w,p)\right)$;
For row of blocks $(i_1,\ldots,i_l) \in \times_{h=1}^{l} \mathcal{R}_p^{(h)}$ lexicographically,
Solve $\boldsymbol{\pi}_{(it+1)}((i_1,\ldots,i_l))Q((i_1,\ldots,i_l),(i_1,\ldots,i_l)) = \mathbf{b}((i_1,\ldots,i_l))$;
For column of blocks $(j_1,\ldots,j_l) \in \times_{h=1}^{l} \mathcal{R}_p^{(h)}$
such that $(j_1,\ldots,j_l) > (i_1,\ldots,i_l)$,
$\mathbf{b}((j_1,\ldots,j_l)) := \mathbf{b}((j_1,\ldots,j_l)) - \omega\boldsymbol{\pi}_{(it+1)}((i_1,\ldots,i_l))Q((i_1,\ldots,i_l),(j_1,\ldots,j_l))$.

To the contrary of BJOR, the nonsingular diagonal blocks $Q((i_1,\ldots,i_l),(i_1,\ldots,i_l))$ 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 $\sum_{p=0}^{N-1} \prod_{h=1}^{l} |\mathcal{R}_p^{(h)}|$ systems directly at each iteration.

Otherwise, one can use an iterative method such as BSOR, since the off-diagonal parts of diagonal blocks are also sums of Kronecker products. After each block is solved for the unknown subvector $\boldsymbol{\pi}_{(it+1)}((i_1,\ldots,i_l))$, $\mathbf{b}(\mathcal{R}_p)$ is updated by multiplying the computed subvector with the corresponding row of blocks above the diagonal.

Note that when $N = 1$, hence, $p = 0$, the two summations on the right-hand side of $\mathbf{b}(\mathcal{R}_p)$ in Algorithm 6 disappear, and since there is a single reachable state space partition in this case, $\mathcal{R}_0$ can be dropped from the argument lists of $\boldsymbol{\pi}_{(it)}$ and $\mathbf{b}$; furthermore, $Q(0,0)_{L(l)}$ becomes $Q_{L(l)}$, implying the fourth term on the right-hand side of $\mathbf{b}$ reduces to $-\omega\boldsymbol{\pi}_{(it)}Q_{L(l)}$ as in Algorithm 2 of [101]. Finally, we emphasize that BSOR at level l reduces to SOR if $Q_{DL(l)} = 0$ (see Remark 4.1 in [315]).

The block iterative solvers, which are sometimes called *two-level* (or *two-stage*) iterative solvers [243], discussed in this section are coded within the `NSolve` package of the APNN toolbox [7, 22]. These solvers are shown to be more effective than point solvers on many test cases [58, 315]. Furthermore, to the contrary of block partitionings considered in [123] 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 the transition matrix of a MC expressed using sums of Kronecker products. Diagonal blocks having identical off-diagonal parts and diagonals which differ by a multiple of the identity [58] are exploited as discussed in Section 4.3. Such diagonal blocks can share and work with the real Schur 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 work in [58] also considers 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 [172] for BGS. Finally, we remark that it is possible to alter the nonzero structure of the transition matrix underlying the Kronecker representation of a MC by reordering factors and states of factors so as to make it more suitable for block iterative methods. Power and JOR methods do not benefit from such reordering since the subdominant eigenvalues of their iteration matrices remain the same.

There are also *Schwarz* methods, which can be considered as a generalization of block iterative methods based on splittings in which the partitioning of the reachable state space $\mathcal{R}$ into N' subsets has overlaps (i.e., $\bigcup_{p=0}^{N'-1}\mathcal{G}_p = \mathcal{R}$ and $\mathcal{G}_p \cap \mathcal{G}_w \neq \varnothing$ for $p \neq w$). Their additive versions [42] become BJOR, and their multiplicative versions become BSOR when the overlaps are removed. Schwarz methods tend to accelerate the convergence of the corresponding block iterative methods, the amount of acceleration depending on the amount of overlap [234]. These methods are yet to be used with Kronecker-based Markovian representations.

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

6.2 Preconditioned Projection Methods

Projection methods for MCs [17, 123, 278, 305] are *non-stationary* iterative methods [168, 280] using a larger number of supplementary vectors than block iterative methods to expedite the solution process. The most commonly used projection methods for the solution of nonsymmetric linear systems are *bi-conjugate gradient* (BCG) [140], *generalized minimal residual* (GMRES) [281], *conjugate gradient squared* (CGS) [299], *quasi-minimal residual* (QMR) [149], and *bi-conjugate gradient stabilized* (BICGSTAB) [317]. Among these, GMRES uses as many supplementary vectors as the Krylov subspace [168, 280] size and therefore has the highest memory requirements.

Projection methods need to be used with preconditioners [28] to result in effective solvers. At each iteration of a preconditioned projection method, the row residual vector, $\mathbf{r}$, is used as the right-hand side of the linear system

$$\mathbf{z}M = \mathbf{r} \tag{6.14}$$

to compute the preconditioned row residual vector, $\mathbf{z}$. 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 (6.10)), 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 (6.14) involving the *preconditioner* matrix, M, should be cheap. It is shown in [123] 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 [123, 279] or explicit *approximate inverses* (AINV) [29]. However, it is still not clear how one can devise ILU- or AINV-type preconditioners for infinitesimal generators that are in the form of (2.10).

So far, various preconditioners are proposed for Kronecker structured representations such as those based on truncated Neumann series [305, 307], the cheap and separable preconditioner [50], circulant preconditioners for a specific class of problems [76], and the Kronecker sum preconditioner [312] which has been shown to work effectively on some small problems. The Kronecker product approximate preconditioner for MCs based on Kronecker products developed in a sequence of papers [210, 211, 212], although encouraging, is in the form of a prototype implementation. Numerical experiments in [50, 52, 211, 212, 307] 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 (6.8). Since

M is based on Kronecker products, a BSOR preconditioner exploiting this property is proposed in [60]. To the contrary of the BSOR preconditioner entertained for sparse MCs in [123], the BSOR preconditioner for MCs based on Kronecker products has a rich structure induced by the lexicographical ordering of states. We provide the BSOR preconditioning step (6.14) using M^{BSOR} at level l for MCs based on Kronecker products in Algorithm 7. You will notice its resemblance to the BSOR iteration *it* at level l for MCs based on Kronecker products in Algorithm 6.

Projection methods and their preconditioned versions discussed in this section are coded within the `NSolve` package of the APNN toolbox [7, 22]. Through numerical experiments, it is shown in [60] 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.

ALGORITHM 7. *BSOR preconditioning step at level l for MCs based on Kronecker products.*

For reachable state space partition $p := 0, \ldots, N-1$,
  $\mathbf{b}(\mathcal{R}_p) := \mathbf{r}(\mathcal{R}_p) - \sum_{w=0}^{p-1} \mathbf{z}(\mathcal{R}_w) Q(w,p)$;
  For row of blocks $(i_1, \ldots, i_l) \in \times_{h=1}^{l} \mathcal{R}_p^{(h)}$ lexicographically,
    Solve $\mathbf{z}((i_1, \ldots, i_l)) Q((i_1, \ldots, i_l), (i_1, \ldots, i_l)) = \mathbf{b}((i_1, \ldots, i_l))$;
    If $\omega \neq 1$,
      $\mathbf{z}((i_1, \ldots, i_l)) := \omega \mathbf{z}((i_1, \ldots, i_l))$;
    For column of blocks $(j_1, \ldots, j_l) \in \times_{h=1}^{l} \mathcal{R}_p^{(h)}$
    such that $(j_1, \ldots, j_l) > (i_1, \ldots, i_l)$,
      $\mathbf{b}((j_1, \ldots, j_l)) := \mathbf{b}((j_1, \ldots, j_l))$
        $-\mathbf{z}((i_1, \ldots, i_l)) Q((i_1, \ldots, i_l), (j_1, \ldots, j_l))$.

It will be interesting to compare JOR, BJOR, and SOR preconditioners as defined in (6.11) and (6.13) with existing preconditoners for MCs based on Kronecker products. Clearly, the class of ML methods proposed in [59] is another candidate for preconditioning projection methods.

In the next section, we introduce a simple version of the ML method [53, 59] for irreducible MCs based on Kronecker products which is intimately related to the well-known IAD method [74, 200, 305], but is not restricted to having two levels. A class of ML methods are then discussed in terms of the simple ML method.

6.3 Multilevel Methods

One of the most effective methods for computing the steady-state vector when the transition matrix of the MC is large and sparse is iterative aggregation–

disaggregation (IAD). To be able to use IAD, a partitioning of the reachable state space as in

$$\mathcal{R} = \bigcup_{p=0}^{N'-1} \mathcal{G}_p, \quad \mathcal{G}_p \cap \mathcal{G}_w = \varnothing \quad \text{for } p \neq w, \tag{6.15}$$

and a function $f : \mathcal{R} \mapsto \mathcal{N}'$ with $\mathcal{N}' = \{0, \ldots, N'-1\}$ that maps the detailed subset of states $\mathcal{G}_p$ to the coarser state $\{p\}$ for $p = 0, \ldots, N'-1$ needs to be identified up to a reordering and renumbering of the states.

Originally the IAD method [289] is devised for DTMCs. For a CTMC, the uniformized generator matrix P in (5.3) (cf. (1.2)) for steady-state analysis may be considered. Given $\boldsymbol{\pi}_{(0)} \in \mathbb{R}_{>0}^{1\times|\mathcal{R}|}$ such that $\boldsymbol{\pi}_{(0)}\mathbf{e} = 1$, IAD proceeds by performing computations at two different levels during iteration it. At the *coarse* level, *aggregation* is performed by mapping $\mathcal{G}_p$ to $\{p\}$ for $p = 0, \ldots, N'-1$, using $\boldsymbol{\pi}_{(it)}$ to define the aggregated system of linear equations

$$\mathbf{z}_{(it)}\, C_{\boldsymbol{\pi}_{(it)}} = \mathbf{z}_{(it)}, \quad \mathbf{z}_{(it)}\mathbf{e} = 1,$$

where

$$C_{\boldsymbol{\pi}_{(it)}} = \begin{pmatrix} c_{\boldsymbol{\pi}_{(it)}}(0,0) & \cdots & c_{\boldsymbol{\pi}^{(it)}}(0,N'-1) \\ \vdots & \ddots & \vdots \\ c_{\boldsymbol{\pi}_{(it)}}(N'-1,0) & \cdots & c_{\boldsymbol{\pi}_{(it)}}(N'-1,N'-1) \end{pmatrix},$$

and then the aggregated linear system is solved for $\mathbf{z}_{(it)} \in \mathbb{R}_{>0}^{1\times N'}$. To this end, the *aggregated* (or *coupling* [241]) *matrix* $C_{\boldsymbol{\pi}_{(it)}}$ needs to be an irreducible and aperiodic transition probability matrix satisfying $C_{\boldsymbol{\pi}_{(it)}} \in \mathbb{R}_{\geqslant 0}^{N'\times N'}$ and $C_{\boldsymbol{\pi}_{(it)}}\mathbf{e} = \mathbf{e}$. At the *fine* level, through *disaggregation* and a splitting-based iterative method, $\mathbf{z}_{(it)}$ is used to compute a hopefully better solution $\boldsymbol{\pi}_{(it+1)}$. The objective of IAD is to converge relatively fast to $\boldsymbol{\pi}$ with a reasonable accuracy.

Now, let us recall a result mentioned in Section 6.1. When none of the diagonal blocks in Q (for that matter, P) can be symmetrically permuted to block diagonal form, $T^{BJacobi}$ and T^{BGS} will have states of each partition in the same periodic class. A consequence of this often overlooked result is that regardless of the periodicity of the iteration matrix T, if such a block iteration is coupled with an aggregation step, *global convergence* of IAD is guaranteed [87].

Here, global convergence refers to the fact that

$$\lim_{it\to\infty} \|\boldsymbol{\pi}_{(it)} - \boldsymbol{\pi}\| = 0 \quad \text{for arbitrary} \quad \boldsymbol{\pi}_{(0)} \in \mathbb{R}_{>0}^{1\times|\mathcal{R}|}.$$

This is a stronger result than *local convergence* which requires that

$$\lim_{it\to\infty} \|\boldsymbol{\pi}_{(it)} - \boldsymbol{\pi}\| = 0 \quad \text{if} \quad \boldsymbol{\pi}_{(0)} \in \Omega(\boldsymbol{\pi}),$$

where $\Omega(\boldsymbol{\pi})$ is some neighborhood of $\boldsymbol{\pi}$.

For fast convergence, *nearly completely decomposable* (NCD) partitionings are considered for DTMCs in [200, 309, 321], which are inspired by earlier work [85, 295] related to NCDness. A block partitioning

$$P = E + \mathrm{diag}(P(0,0), \ldots, P(N'-1, N'-1))$$

is said to be NCD if $\|E\|_1$, that is, the *degree of coupling*, is relatively small with respect to 1. NCD partitionings are frequently met in practice due to the existence of different time scales in system models [308]. An NCD partitioning for a user-specified *decomposability parameter* $\gamma \in (0,1)$ can be obtained as discussed in [97]. When the block partitioning is NCD, $\mathbf{z}_{(it)}$ may need to be computed with the help of the *Grassmann–Taksar–Heyman (GTH)* idea [167, 293] which uses only positive floating-point arithmetic to achieve high accuracy as shown in [121]. That NCD partitioning-based IAD with BGS as the block iteration converges globally with *convergence factor* $\|E\|_1$ (where rate of convergence is the negated logarithm of convergence factor) [74, 237, 304] provides a very strong result for DTMCs. However, the block partitioning suggested by the Kronecker form may not be NCD. Therefore, IAD methods using non-NCD partitionings possibly with point iterative methods based on splittings need to be considered [77, 176, 182, 204, 244, 246, 316, 325]. This is something we investigate next so that it guides us in developing the ML method for Kronecker-based Markovian representations.

The local convergence proof of an IAD method for nonnegative matrices that uses the power method after disaggregation is provided in [221]. The particular IAD method computes the nonnegative solution vector of a problem that is intimately related to (6.9), but has a nonnegative consistent right-hand side vector, and for a given weight vector $\mathbf{w} \in \mathbb{R}_{>0}^{|\mathcal{R}|\times 1}$, the weighted ith row sum of the matrix at hand is less than $\mathbf{w}(i)$ for $i = 0, \ldots, |\mathcal{R}| - 1$. The local convergence result of the IAD method in [221] is extended to DTMCs in [222] and to inexact solution of the aggregated system in [231].

The *aggregation* (or *restriction*) *operator* $R \in \mathbb{R}_{\geqslant 0}^{|\mathcal{R}|\times N'}$ and the *disaggregation* (or *prolongation*) *operator* $S_{\boldsymbol{\pi}_{(it)}} \in \mathbb{R}_{\geqslant 0}^{N'\times|\mathcal{R}|}$ used in the IAD method for DTMCs are given entrywise in [224] as

$$r(i,p) = \begin{cases} \mathbf{w}(i) & \text{if } f(i) = p \\ 0 & \text{otherwise} \end{cases},$$

$$s_{\boldsymbol{\pi}_{(it)}}(p,i) = \begin{cases} \dfrac{\boldsymbol{\pi}_{(it)}(i)}{\sum_{j\in\mathcal{R},\ f(j)=p} \mathbf{w}(j)\boldsymbol{\pi}_{(it)}(j)} & \text{if } f(i) = p \\ 0 & \text{otherwise} \end{cases}$$

with a weight vector $\mathbf{w} \in \mathbb{R}_{>0}^{|\mathcal{R}|\times 1}$ as in [221] normally set to $\mathbf{e}$.

The disaggregation operator $S_{\boldsymbol{\pi}_{(it)}}$ depends on the current solution vector $\boldsymbol{\pi}_{(it)} \in \mathbb{R}_{>0}^{1\times|\mathcal{R}|}$ and has the same nonzero structure as R^T, implying

$$S_{\boldsymbol{\pi}_{(it)}} R = I.$$

When $\boldsymbol{\pi}_{(it)} \in \mathbb{R}_{>0}^{1\times|\mathcal{R}|}$, it is also possible to define the nonnegative *projector*

$$B_{\boldsymbol{\pi}_{(it)}} = RS_{\boldsymbol{\pi}_{(it)}}$$

which satisfies

$$B_{\boldsymbol{\pi}_{(it)}}^2 = B_{\boldsymbol{\pi}_{(it)}}, \quad \boldsymbol{\pi}_{(it)}B_{\boldsymbol{\pi}_{(it)}} = \boldsymbol{\pi}_{(it)}, \quad \text{and} \quad B_{\boldsymbol{\pi}_{(it)}}\mathbf{w} = \mathbf{w}.$$

In terms of the aggregation and disaggregation operators, the aggregation step of IAD for DTMCs computes

$$C_{\boldsymbol{\pi}_{(it)}} := S_{\boldsymbol{\pi}_{(it)}}PR$$

and solves for $\mathbf{z}_{(it)} \in \mathbb{R}_{>0}^{1\times N'}$ in

$$\mathbf{z}_{(it)}C_{\boldsymbol{\pi}_{(it)}} = \mathbf{z}_{(it)}, \quad \mathbf{z}_{(it)}\mathbf{e} = 1,$$

whereas the disaggregation step followed by one power iteration is given as

$$\boldsymbol{\pi}_{(it+1)} := \mathbf{z}_{(it)}S_{\boldsymbol{\pi}_{(it)}}T \quad \text{with} \quad T = P.$$

A more general IAD method to compute the steady-state vector of DTMCs is provided in [225]. The method therein is shown to be globally convergent if a sufficiently high power of P is used in the aggregation step or a sufficiently large number of power iterations are performed on $\hat{P} = 0.5(I + P)$ after the disaggregation step. This IAD method is investigated further in a sequence of papers [223, 226] together with another intimately related IAD method devised for the singular version of the problem with the right-hand side in [221]. Letting

$$\Pi = \mathbf{e}\boldsymbol{\pi}$$

denote the square matrix with the steady-state vector $\boldsymbol{\pi}$ in its rows, that is, the part of P that corresponds to the eigenvalue 1, and using the *spectral decomposition* of P as in

$$P = \Pi + Z,$$

so that

$$\Pi^2 = \Pi, \quad \Pi Z = Z\Pi = 0, \quad \rho(Z) < 1, \quad \text{and} \quad \Pi(P - I) = 0,$$

it is shown that both IAD methods are locally convergent if a particular convergent matrix (dependent on the nonnegative projector $B_{\boldsymbol{\pi}_{(it)}}$, the iteration matrix T of the method used after disaggregation, and Z) exists. Furthermore, fast convergence of these IAD methods in one iteration is guaranteed when off-diagonal blocks in the strictly block lower-triangular part of T are outer products and therefore rank–1 [223].

The version of the IAD method with one power iteration taking place after disaggregation is shown in [228] to be locally convergent for $\delta \in (0, 1)$ with

convergence factor $(1-\delta)$ when $P \geqslant \delta\Pi$, or with convergence factor $\sqrt{1-\delta}$ when P has at least one positive column with 1-norm δ. Note that the former condition requires $P > 0$; therefore, both conditions are difficult to meet in practice. The same IAD method with P^{η_2} used in the aggregation step, η_1 power iterations (i.e., $T = P^{\eta_1}$) after the disaggregation step, and $\eta_1 \geqslant \eta_2$ is globally convergent for $\delta \in (0,1)$ when

$$P^{\eta_2} \geqslant \delta\Pi \quad \text{and} \quad \eta_1 \geqslant \eta_2 \frac{ln\delta - ln2}{ln(1-\delta)}.$$

We remark that since the power method is globally convergent for $\delta \in (0,1)$ with convergence factor $(1-\delta)$ when $P \geqslant \delta\Pi$, the IAD method with one power iteration taking place after disaggregation is not better than the power method and, therefore, cannot be recommended. Furthermore, IAD methods that are based on using powers of P during the aggregation step do not seem to be feasible in a sparse setting due to new nonzeros that would be introduced by this operation.

A class of IAD methods which use P in the aggregation step and a polynomial of P as the iteration matrix T of the method after disaggregation is considered in [269]. Therein, it is shown that IAD is locally convergent when $T = P$ and $p(i,i) > 0$ for $i = 0, \ldots, |\mathcal{R}| - 1$, or when $T = \alpha P + (1-\alpha)I$ for $\alpha \in (0,1)$, implying $T(i,i) > 0$ for $i = 0, \ldots, |\mathcal{R}| - 1$. In the latter case, the convergence factor is minimal for $\alpha \in (0.5, 1)$. In [270], the aperiodicity of an irreducible transition probability matrix associated with the *stochastic complement* [241] of $B_{\boldsymbol{\pi}_{(it)}}(p,p)P(p,p)$ for $p = 0, \ldots, N' - 1$ is provided as a necessary and sufficient condition for local convergence of IAD with one power iteration after disaggregation. The convergence factor of IAD for subvector p of $\boldsymbol{\pi}$ turns out to be the magnitude of the subdominant eigenvalue of this matrix. Many other interesting results are derived from experiments in [227, 271].

In the Kronecker-based representation of MCs, the partitioning (6.15) to be used by the IAD method can very well correspond to the block partitioning (2.10) at level 0 for which $N' = N$ and $\mathcal{G}_p = \mathcal{R}_p$ for $p = 0, \ldots, N-1$ when there are multiple reachable state space partitions (i.e., $N > 1$), or to another block partitioning at a higher level number (see, for instance, those in Table 4.1). It is even possible to use different level numbers at each reachable state space partition $\mathcal{R}_p$ to control the sizes of the diagonal blocks underlying the partitioning.

The challenge is to provide a scalable implementation of IAD. The first results along this direction for Kronecker-based representations of MCs couple aggregation–disaggregation steps with various iterative methods to accelerate convergence [43, 49, 50, 52]. 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 [47] and [48], respectively. The adaptive IAD method in [48] is improved in [53] through a recursive definition and called ML. Here, we first present this method for generator matrices in the Kronecker form of (2.10).

The ML method is conceived so that aggregation of states takes place in a systematic manner, within each reachable state space partition independently of other reachable state space partitions. To keep the discussion simple, we assume that aggregation is performed in the order of submodel indices from 1 to H and disaggregation in the reverse order. At level 0, we have the unaggregated generator matrix Q. Therefore, at level 0 we aggregate submodel 1 and then move to level 1. At level l, we aggregate submodel $l+1$ since submodels 1 through l will have been aggregated previously. If we forget about the diagonal correction that sums the rows of the generator matrix to 0, term $k \in \mathcal{K}_{p,w}$ in aggregated block (p, w) at level l will correspond to the aggregation of the first $l+1$ submodels. In this way, it is possible to define an aggregated matrix at each partitioning level l for $l = 1, \ldots, H$ that is an $(N \times N)$ block matrix. Block (p, w) of this aggregated matrix can be expressed as a sum of Kronecker products with $|\mathcal{K}_{p,w}|$ terms as in (2.10) for $p, w = 0, \ldots, N-1$. Thus, it will be represented compactly by the multiplication of α_k with a diagonal $(\prod_{h=l+2}^{H} |\mathcal{R}_p^{(h)}| \times \prod_{h=l+2}^{H} |\mathcal{R}_p^{(h)}|)$ matrix describing the effect of aggregating the first $l+1$ submodels on the remaining $H-l-1$ unaggregated submodels, which in turn is multiplied by the Kronecker product $\bigotimes_{h=l+2}^{H} Q_k^{(h)}(\mathcal{R}_p^{(h)}, \mathcal{R}_w^{(h)})$ corresponding to the unaggregated $H-l-1$ submodels.

This approach requires us to first associate with the H-tuple state $\mathbf{i} \in \mathcal{R}_p$ the partition number p to which it belongs. We need this simply because, after aggregation of a particular submodel, it is possible for states in different reachable state space partitions to map to the same aggregated state; but, as motivated in the previous paragraph, each aggregated state must remain in its original partition. This can be done as in [61] by inserting the partition number p as the $(H+1)$st entry at the end of the H-tuple representing state $\mathbf{i} \in \mathcal{R}_p$ so that states are represented by $(H+1)$-tuples or, for instance, by using a subscript p with $\mathbf{i} \in \mathcal{R}_p$ as we will do here. Second, a level number needs to be associated with reachable state space partition $\mathcal{R}_p$ and the aggregated states $\mathbf{i} \in \mathcal{R}_p$ for $p = 0, \ldots, N-1$ to specify the aggregated operator using sums of Kronecker products at each level.

So, let $\mathbf{i}_{(p,l)} \in \mathcal{R}_{p,l}$, where

$$\mathcal{R}_{p,l} = \bigtimes_{h=l+1}^{H} \mathcal{R}_p^{(h)} \quad \text{for } p = 0, \ldots, N-1 \text{ and } l = 0, \ldots, H-1$$

denotes the aggregated reachable state space partition p at level l in which submodels 1 through l are aggregated, with

$$\mathcal{R}_{p,0} = \mathcal{R}_p \quad \text{and} \quad \mathcal{R}_{p,H} = \{p\},$$

and the mapping

$$f_{(p,l)} : \mathcal{R}_{p,l} \mapsto \mathcal{R}_{p,l+1}$$

represents the aggregation of submodel $l+1$ with state space $\mathcal{R}_p^{(l+1)}$ for $p = 0, \ldots, N-1$ so that state $\mathbf{i}_{(p,l)} \in \mathcal{R}_{p,l}$ is mapped to state $\mathbf{i}_{(p,l+1)} \in \mathcal{R}_{p,l+1}$.

Furthermore, let the aggregated generator matrices $\tilde{Q}_{(it,l)}$ with aggregated reachable state space partitions $\mathcal{R}_{p,l}$, $p = 0, \ldots, N-1$, be defined at levels $l = 1, \ldots, H$ with $\tilde{Q}_{(it,0)} = Q$ for iteration it. Observe that $\tilde{Q}_{(it,l)}$ is an $(\sum_{p=0}^{N-1} |\mathcal{R}_{p,l}| \times \sum_{p=0}^{N-1} |\mathcal{R}_{p,l}|)$ matrix. Finally, let the power method be used as *smoother* (or *accelerator*) before aggregation $\eta_{(it,l)}$ times and after disaggregation $\nu_{(it,l)}$ times with the uniformization parameter

$$\Gamma_{(it,l)} \in \Big[\max_{\mathbf{i}_{(p,l)} \in \mathcal{R}_{p,l},\ p=0,\ldots,N-1} |\tilde{q}_{(it,l)}(\mathbf{i}_{(p,l)}, \mathbf{i}_{(p,l)})|, \infty \Big)$$

at level l for iteration it.

Then the ML iteration matrix that facilitates the ML iteration

$$\boldsymbol{\pi}_{(it+1,l)} := \boldsymbol{\pi}_{(it,l)} T^{ML}_{(it,l)} \quad \text{for } it = 0, \ldots, \texttt{maxit} - 1$$

at level l for iteration it is given by

$$T^{ML}_{(it,l)} = \left(I + \frac{1}{\Gamma_{(it,l)}} \tilde{Q}_{(it,l)} \right)^{\eta_{(it,l)}} R_{(l)} T^{ML}_{(it,l+1)} S_{\mathbf{x}_{(it,l)}} \left(I + \frac{1}{\Gamma_{(it,l)}} \tilde{Q}_{(it,l)} \right)^{\nu_{(it,l)}}. \tag{6.16}$$

Note that the definition of $T^{ML}_{(it,l)}$ is recursive, and to the contrary of block iterative methods, the ML iteration matrix in (6.16) changes from iteration to iteration, and hence, the method is non-stationary.

At iteration it, the recursion ends and backtracking starts when $\tilde{Q}_{(it,l+1)}$ is the last aggregated generator matrix and solved to give

$$T^{ML}_{(it,l+1)} = \mathbf{e}\boldsymbol{\pi}_{(it+1,l+1)}, \quad \text{where } \boldsymbol{\pi}_{(it+1,l+1)} \tilde{Q}_{(it,l+1)} = \mathbf{0} \text{ and } \boldsymbol{\pi}_{(it+1,l+1)} \mathbf{e} = 1.$$

The level to end recursion depends on available memory since there must be space to store and factorize the aggregated generator matrix at that level if a direct method is employed for an accurate solution. When Q is irreducible and $\boldsymbol{\pi}_{(0,0)} \in \mathbb{R}^{1 \times |\mathcal{R}|}_{>0}$, the aggregated generator matrices $\tilde{Q}_{(it,l+1)}$ are irreducible [53, 61], and the ML method has been observed to converge if a sufficient number of smoothings are performed to improve the approximate solution vector $\boldsymbol{\pi}_{(it,l)}$ at each level.

Clearly, the implementation of the ML method with the iteration matrix in (6.16) should not require the explicit generation and storage of the aggregated generator matrices $\tilde{Q}_{(it,l)}$ for MCs based on Kronecker products. To the contrary, as other iterative methods before, it should rely on vector–Kronecker product multiplications with smaller matrices and some vector operations.

The pre-smoothed vector in (6.16) is obtained from

$$\mathbf{x}_{(it,l)} := \boldsymbol{\pi}_{(it,l)} \left(I + \frac{1}{\Gamma_{(it,l)}} \tilde{Q}_{(it,l)} \right)^{\eta_{(it,l)}}. \tag{6.17}$$

The aggregation operator $R_{(l)} \in \mathbb{R}_{\geqslant 0}^{\sum_{p=0}^{N-1} |\mathcal{R}_{p,l}| \times \sum_{p=0}^{N-1} |\mathcal{R}_{p,l+1}|}$ and the disaggregation operator $S_{\mathbf{x}_{(m,l)}} \in \mathbb{R}^{\sum_{p=0}^{N-1} |\mathcal{R}_{p,l+1}| \times \sum_{p=0}^{N-1} |\mathcal{R}_{p,l}|}$ are given entrywise in [61] as

$$r_{(l)}(\mathbf{i}_{(p,l)}, \mathbf{i}_{(p,l+1)}) = \begin{cases} 1 & \text{if } f_{(p,l)}(\mathbf{i}_{(p,l)}) = \mathbf{i}_{(p,l+1)} \\ 0 & \text{otherwise} \end{cases} \tag{6.18}$$

and

$$\begin{aligned} s_{\mathbf{x}_{(it,l)}}(\mathbf{i}_{(p,l+1)}, \mathbf{i}_{(p,l)}) &= \frac{\mathbf{x}_{(it,l)}(\mathbf{i}_{(p,l)})}{\sum\limits_{\mathbf{j}_{(p,l)} \in \mathcal{R}_{p,l},\ f_{(p,l)}(\mathbf{j}_{(p,l)}) = \mathbf{i}_{(p,l+1)}} \mathbf{x}_{(it,l)}(\mathbf{j}_{(p,l)})} \\ &\qquad \text{if } f_{(p,l)}(\mathbf{i}_{(p,l)}) = \mathbf{i}_{(p,l+1)}, \\ s_{\mathbf{x}_{(it,l)}}(\mathbf{i}_{(p,l+1)}, \mathbf{i}_{(p,l)}) &= 0 \quad \text{otherwise} \end{aligned} \tag{6.19}$$

$$\text{for } \mathbf{i}_{(p,l)} \in \mathcal{R}_{p,l},\ \mathbf{i}_{(p,l+1)} \in \mathcal{R}_{p,l+1},\ \text{and } p = 0, \ldots, N-1\ .$$

Observe that both operators at level l are $(N \times N)$ block diagonal due to our choice to aggregate states only within each partition $\mathcal{R}_{p,l}$ for $p = 0, \ldots, N-1$.

The aggregation operator $R_{(l)}$ in (6.18) is defined by $f_{(p,l)} : \mathcal{R}_{p,l} \mapsto \mathcal{R}_{p,l+1}$ for $p = 0, \ldots, N-1$, and therefore is constant and need not be stored. At level l, the $|\mathcal{R}_{p,l}| = \prod_{h=l+1}^{H} |\mathcal{R}_p^{(h)}|$ states represented by $(H-l)$-tuples are mapped to the $|\mathcal{R}_{p,l+1}| = \prod_{h=l+2}^{H} |\mathcal{R}_p^{(h)}|$ states represented by $(H-l-1)$-tuples by $f_{(p,l)}$ through aggregation of the leading dimension $\mathcal{R}_p^{(l+1)}$ in $\mathcal{R}_{p,l}$ for $p = 0, \ldots, N-1$. We remark that this corresponds to an aggregation based on a contiguous and non-interleaved block partitioning if the states in $\mathcal{R}_{p,l}$ were ordered anti-lexicographically. On the other hand, the disaggregation operator $S_{\mathbf{x}_{(it,l)}}$ in (6.19) depends on the smoothed vector $\mathbf{x}_{(it,l)}$ in (6.17) and has the nonzero structure of $R_{(l)}^T$. Therefore, $S_{\mathbf{x}_{(it,l)}}$ can be stored in a vector of length $\sum_{p=0}^{N-1} |\mathcal{R}_{p,l}|$ since it has one nonzero per column by definition. These vectors amount to a total storage of $\sum_{l=0}^{H-1} \sum_{p=0}^{N-1} |\mathcal{R}_{p,l}|$ floating-point values if the recursion terminates at level H.

The aggregation operator $R_{(l)}$ and the disaggregation operator $S_{\mathbf{x}_{(it,l)}}$ have the same properties of the two operators used in the IAD method for DTMCs in this section. Similarly, it is also possible to define the nonnegative projector $B_{\mathbf{x}_{(it,l)}} \in \mathbb{R}_{\geqslant 0}^{\sum_{p=0}^{N-1} |\mathcal{R}_{p,l}| \times \sum_{p=0}^{N-1} |\mathcal{R}_{p,l}|}$ at level l as

$$B_{\mathbf{x}_{(it,l)}} = R_{(l)} S_{\mathbf{x}_{(it,l)}} \quad \text{for } l = 0, \ldots, H-1.$$

In the ML method, the pre-smoothed vector is aggregated using

$$\boldsymbol{\pi}_{(it,l+1)} := \mathbf{x}_{(it,l)} R_{(l)} \tag{6.20}$$

and passed to level $l+1$, which has the aggregated generator matrix

$$\tilde{Q}_{(it,l+1)} = S_{\mathbf{x}_{(it,l)}} \tilde{Q}_{(it,l)} R_{(l)}\ . \tag{6.21}$$

In [61], it is shown that $\tilde{Q}_{(it,l+1)}$ is a block matrix that can be expressed using sums of Kronecker products as in (2.10) with $\sum_{p=0}^{N-1}\sum_{w=0}^{N-1}|\mathcal{K}_{p,w}|$ *aggregation vectors* named $\mathbf{a}$ each of length at most $\max_{p\in\{0,\ldots,N-1\}}(\prod_{h=l+2}^{H}|\mathcal{R}_p^{(h)}|)$ and the submatrices corresponding to the factors $(l+2)$ through H.

More specifically, the $\mathbf{i}_{(p,l+1)}$st entry of the aggregation vector corresponding to the kth term in the Kronecker representation at level $(l+1)$ for block (p,w) at iteration it with $\mathbf{i}_{(p,l+1)} \in \mathcal{R}_{p,l+1}$ and $k \in \mathcal{K}_{p,w}$ is computed as

$$\begin{aligned}&\mathbf{a}_{(it,l+1,p,w,k)}(\mathbf{i}_{(p,l+1)}) \qquad (6.22)\\ &:= \frac{\sum_{\mathbf{j}_{(p,l)}\in\mathcal{R}_{p,l},\ f_{(p,l)}(\mathbf{j}_{(p,l)})=\mathbf{i}_{(p,l+1)}} \mathbf{x}_{(it,l)}(\mathbf{j}_{(p,l)})\ \mathbf{a}_{(it,l,p,w,k)}(\mathbf{j}_{(p,l)})\ row\_sum}{\boldsymbol{\pi}_{(it,l+1)}(\mathbf{i}_{(p,l+1)})},\end{aligned}$$

where

$$row\_sum := \mathbf{e}^T_{\mathbf{j}_{(p,l)}(l+1)} Q_k^{(l+1)}(\mathcal{R}_p^{(l+1)}, \mathcal{R}_w^{(l+1)})\mathbf{e}$$

yields the sum of entries in row $\mathbf{j}_{(p,l)}(l+1) \in \mathcal{R}_p^{(l+1)}$ of $Q_k^{(l+1)}(\mathcal{R}_p^{(l+1)}, \mathcal{R}_w^{(l+1)})$ with $\mathbf{e}_{\mathbf{j}_{(p,l)}(l+1)}$ being the $\mathbf{j}_{(p,l)}(l+1)$st column of I of order $|\mathcal{R}_p^{(l+1)}|$. At level 0, we set

$$\mathbf{a}_{(it,0,p,w,k)} := \mathbf{e}.$$

Then block (p,w) of $\tilde{Q}_{(it,l+1)}$ for $p,w = 0,\ldots,N-1$ can be expressed as

$$\tilde{Q}_{(it,l+1)}(p,w) = \begin{cases} \sum_{k\in\mathcal{K}_{p,w}} \tilde{Q}_{(it,l+1,k)}(p,w) + \tilde{Q}_{(it,l+1,D)}(p,p) & \text{if } p = w \\ \sum_{k\in\mathcal{K}_{p,w}} \tilde{Q}_{(it,l+1,k)}(p,w) & \text{otherwise} \end{cases},$$

where

$$\begin{aligned}\tilde{Q}_{(it,l+1,k)}(p,w) &= \alpha_k\ \mathrm{diag}(\mathbf{a}_{(it,l+1,p,w,k)}) \bigotimes_{h=l+2}^{H} Q_k^{(h)}(\mathcal{R}_p^{(h)}, \mathcal{R}_w^{(h)}),\\ \tilde{Q}_{(it,l+1,D)}(p,p) &= -\sum_{w=0}^{N-1}\sum_{k\in\mathcal{K}_{p,w}} \alpha_k\ \mathrm{diag}(\mathbf{a}_{(it,l+1,p,w,k)}) \\ &\qquad \bigotimes_{h=l+2}^{H} \mathrm{diag}(Q_k^{(h)}(\mathcal{R}_p^{(h)}, \mathcal{R}_w^{(h)})\mathbf{e}). \qquad (6.23)\end{aligned}$$

Observe that $\tilde{Q}_{(it,l+1,D)}(p,p)$ returns a diagonal matrix with negative diagonal entries so that

$$\sum_{w=0}^{N-1} \tilde{Q}_{(it,l+1)}(p,w)\mathbf{e} = \mathbf{0} \quad \text{for } p = 0,\ldots,N-1.$$

In other words, row sums of the row of blocks corresponding to aggregated reachable state space partition p at level $l+1$, $\mathcal{R}_{p,l+1}$, of $\tilde{Q}_{(it,l+1)}$ must be equal to the $\mathbf{0}$ vector. If the recursion ends at level H, then $\tilde{Q}_{(it,H)}$ is an $(N \times N)$ generator matrix and therefore is generated and stored explicitly in row sparse format when there are multiple reachable state space partitions (i.e., $N > 1$) so that it can be solved either directly (e.g., by Gaussian elimination) if N is small, else iteratively using the smoother and the current approximation $\boldsymbol{\pi}_{(it,H)}$ as the starting vector. When $N = 1$, $\tilde{Q}_{(it,H)} = 0$, and hence, $\boldsymbol{\pi}_{(it+1,H)}$ can be set to 1 to start backtracking from the recursion.

The aggregation vectors $\mathbf{a}_{(it,l,p,w,k)}$ for $k \in \mathcal{K}_{p,w}$ at $l = 0$ by definition consist of all 1's, and therefore need not be stored. Furthermore, the computation of $\mathbf{a}_{(it,l+1,p,w,k)}$ in (6.22) suggests that if $\mathbf{a}_{(it,l,p,w,k)} = \mathbf{e}$ and *row_sum* evaluates to 1 for all $\mathbf{i}_{(p,l+1)} \in \mathcal{R}_{p,l+1}$, then $\mathbf{a}_{(it,l+1,p,w,k)} = \mathbf{e}$, since in this case the summation in the numerator will evaluate to the value in the denominator by the definition of the aggregation operator $R_{(l)}$. This is possible, for instance, when $Q_k^{(h)}(\mathcal{R}_p^{(h)}, \mathcal{R}_w^{(h)})$ for $h = 1, \ldots, l+1$ are all diagonal matrices of size $(|\mathcal{R}_p^{(h)}| \times |\mathcal{R}_w^{(h)}|)$ with 1's along their diagonal. Since submodel matrices forming $\tilde{Q}_{(it,l+1)}(p,w)$ for $p \neq w$ can very well be rectangular, we refrain from using I and remark that such aggregation vectors need not be stored either. Savings are also possible for those $k \in \mathcal{K}_{p,p}$ that have a single $Q_k^{(h)}(\mathcal{R}_p^{(h)}, \mathcal{R}_p^{(h)}) \neq I$ for $h = 1, \ldots, H$. In this case, the contribution of $\mathbf{a}_{(it,l+1,p,p,k)}$ can only be to the diagonal of $\tilde{Q}_{(it,l+1)}(p,p)$ and its effect will be canceled with the corresponding diagonal correction due to $\tilde{Q}_{(it,l+1,D)}(p,p)$. This implies that setting $\mathbf{a}_{(it,l+1,p,p,k)} = \mathbf{e}$ and not storing it will not change the result in this case as well.

The aggregation vectors for block (p,w) at a particular level have the same length but vary in length from $\prod_{h=2}^{H} |\mathcal{R}_p^{(h)}|$ at level 1 to $|\mathcal{R}_p^{(H)}|$ at level $(H-1)$, implying a storage requirement of at most

$$\sum_{p=0}^{N-1} \sum_{w=0}^{N-1} |\mathcal{K}_{p,w}| \sum_{l=1}^{H-1} \prod_{h=l+1}^{H} |\mathcal{R}_p^{(h)}|$$

floating-point values to facilitate the Kronecker representation of aggregated generator matrices. We remark that grouping of factors will further reduce the storage requirement for aggregation vectors.

Example 2. (ctnd.) Consider our three-dimensional problem with the parameter set

$$(\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5, \alpha_6) = (\lambda_1, \lambda_2, \lambda_3, \mu_1, \mu_2, \mu_3) = (1, 2, 3, 2, 4, 6),$$

the initial distribution $\boldsymbol{\pi}_{(0,0)} = \mathbf{e}/12$, $\Gamma_{(0,0)} = 14$, and $\eta_{(0,0)} = \nu_{(0,0)} = 1$. Since

$$\tilde{Q}_{(0,0)} = \begin{array}{c} \\ 0\,0\,0 \\ 0\,0\,1 \\ 1\,0\,0 \\ 1\,0\,1 \\ 2\,0\,0 \\ 2\,0\,1 \\ 0\,1\,0 \\ 0\,1\,1 \\ 1\,1\,0 \\ 1\,1\,1 \\ 0\,0\,2 \\ 1\,0\,2 \end{array} \begin{array}{c} \begin{array}{cccccccccccc} 0 & 0 & 1 & 1 & 2 & 2 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 2 & 2 \end{array} \\ \left(\begin{array}{cccccccccccc} -9 & 6 & 1 & & & & 2 & & & & & \\ 6 & -12 & & 1 & & & & 2 & & & 3 & \\ 2 & & -11 & 6 & 1 & & & & 2 & & & \\ & 2 & 6 & -14 & & 1 & & & & 2 & & 3 \\ \hline & & 2 & & -2 & & & & & & & \\ & & & 2 & 6 & -8 & & & & & & \\ \hline 4 & & & & & & -4 & & & & & \\ & 4 & & & & & 6 & -10 & & & & \\ & & 4 & & & & 2 & & -6 & & & \\ & & & 4 & & & & 2 & 6 & -12 & & \\ \hline & 6 & & & & & & & & & -6 & \\ & & & 6 & & & & & & & 2 & -8 \end{array}\right) \end{array},$$

$\mathbf{x}_{(0,0)} = \boldsymbol{\pi}_{(0,0)}(I + \tilde{Q}_{(0,0)}/14)$ from (6.17) yields

$$\mathbf{x}_{(0,0)} = \left(\frac{17}{168}, \frac{20}{168}, \frac{16}{168}, \frac{19}{168}, \frac{19}{168}, \frac{7}{168}, \frac{20}{168}, \frac{8}{168}, \frac{16}{168}, \frac{4}{168}, \frac{13}{168}, \frac{9}{168}\right).$$

In this example, there are $N = 4$ reachable state space partitions, and hence, the aggregation operator in (6.18) and the disaggregation operator in (6.19) are block diagonal matrices. At level 0, they are, respectively, given by

$$R_{(0)} = \begin{array}{c} \\ 0\,0\,0 \\ 0\,0\,1 \\ 1\,0\,0 \\ 1\,0\,1 \\ 2\,0\,0 \\ 2\,0\,1 \\ 0\,1\,0 \\ 0\,1\,1 \\ 1\,1\,0 \\ 1\,1\,1 \\ 0\,0\,2 \\ 1\,0\,2 \end{array} \begin{array}{c} \begin{array}{cccccccc} 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 2 \end{array} \\ \left(\begin{array}{cccccccc} 1 & & & & & & \\ & 1 & & & & & \\ 1 & & & & & & \\ & 1 & & & & & \\ \hline & & 1 & & & & \\ & & & 1 & & & \\ \hline & & & & 1 & & \\ & & & & & 1 & \\ & & & & 1 & & \\ & & & & & 1 & \\ \hline & & & & & & 1 \\ & & & & & & 1 \end{array}\right) \end{array}$$

and

$$S_{\mathbf{x}_{(0,0)}} = \begin{array}{c} \\ 0\,0 \\ 0\,1 \\ 0\,0 \\ 0\,1 \\ 1\,0 \\ 1\,1 \\ 0\,2 \end{array} \begin{array}{c} \begin{array}{cccccccccccc} 0 & 0 & 1 & 1 & 2 & 2 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 2 & 2 \end{array} \\ \left(\begin{array}{cccccccccccc} \frac{17}{33} & & \frac{16}{33} & & & & & & & & & \\ & \frac{20}{39} & & \frac{19}{39} & & & & & & & & \\ & & & & 1 & & & & & & & \\ & & & & & 1 & & & & & & \\ & & & & & & \frac{20}{36} & & \frac{16}{36} & & & \\ & & & & & & & \frac{8}{12} & & \frac{4}{12} & & \\ & & & & & & & & & & \frac{13}{22} & \frac{9}{22} \end{array}\right) \end{array}.$$

The 12 states represented by 3-tuples in $\mathcal{R}$ are mapped to the 7 states represented by 2-tuples in $\mathcal{R}_{0,1}$, $\mathcal{R}_{1,1}$, $\mathcal{R}_{2,1}$, and $\mathcal{R}_{3,1}$. For instance, states $(0,0,0)$ and $(1,0,0)$ in $\mathcal{R}_{0,0}$ are mapped to $(0,0)$ in $\mathcal{R}_{0,1}$, whereas states $(0,0,2)$ and $(1,0,2)$ in $\mathcal{R}_{3,0}$ are mapped to $(0,2)$ in $\mathcal{R}_{3,1}$. Note that in this example, aggregation of submodel 1 at level 0 does not yield a reduction in the number of states in reachable state space partition $\mathcal{R}_{1,1}$ because $|\mathcal{R}_1^{(1)}| = 1$.

Using $R_{(0)}$ in (6.20), we obtain the starting approximation at level 1 as

$$\boldsymbol{\pi}_{(0,1)} = \left(\frac{33}{168}, \frac{39}{168}, \frac{19}{168}, \frac{7}{168}, \frac{36}{168}, \frac{12}{168}, \frac{22}{168}\right).$$

Through the interaction matrix

$$\begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\ \left(\begin{array}{cccc} \{1,3,4,6\} & \{1\} & \{2\} & \{3\} \\ \{4\} & \{6\} & & \\ \{5\} & & \{4,6\} & \\ \{6\} & & & \{4\} \end{array}\right) \end{array},$$

(cf. (2.8)), mapping of submodel states to reachable state space partitions given in Table 2.1, submodel 1's transition submatrices

$$Q_1^{(1)}(\mathcal{R}_0,\mathcal{R}_0) = \begin{pmatrix} & 1 \end{pmatrix}, \quad Q_3^{(1)}(\mathcal{R}_0,\mathcal{R}_0) = I_2, \quad Q_4^{(1)}(\mathcal{R}_0,\mathcal{R}_0) = \begin{pmatrix} & \\ 1 & \end{pmatrix},$$

$$Q_6^{(1)}(\mathcal{R}_0,\mathcal{R}_0) = I_2, \quad Q_1^{(1)}(\mathcal{R}_0,\mathcal{R}_1) = \begin{pmatrix} \\ 1 \end{pmatrix}, \quad Q_2^{(1)}(\mathcal{R}_0,\mathcal{R}_2) = I_2,$$

$$Q_3^{(1)}(\mathcal{R}_0,\mathcal{R}_3) = I_2, \quad Q_4^{(1)}(\mathcal{R}_1,\mathcal{R}_0) = \begin{pmatrix} & 1 \end{pmatrix}, \quad Q_6^{(1)}(\mathcal{R}_1,\mathcal{R}_1) = I_1,$$

$$Q_5^{(1)}(\mathcal{R}_2,\mathcal{R}_0) = I_2, \quad Q_4^{(1)}(\mathcal{R}_2,\mathcal{R}_2) = \begin{pmatrix} & \\ 1 & \end{pmatrix}, \quad Q_6^{(1)}(\mathcal{R}_2,\mathcal{R}_2) = I_2,$$

$$Q_6^{(1)}(\mathcal{R}_3,\mathcal{R}_0) = I_2, \quad Q_4^{(1)}(\mathcal{R}_3,\mathcal{R}_3) = \begin{pmatrix} & \\ 1 & \end{pmatrix},$$

and the definition of aggregation vectors in (6.22), the 14 vectors used to represent the aggregated generator matrix $\tilde{Q}_{(0,1)}$ at level 1 are computed as

$$\mathbf{a}_{(0,1,0,0,1)} = \left(\frac{17}{33}, \frac{20}{39}\right)^T, \quad \mathbf{a}_{(0,1,0,0,4)} = \mathbf{a}_{(0,1,0,1,1)} = \left(\frac{16}{33}, \frac{19}{39}\right)^T,$$

$$\mathbf{a}_{(0,1,2,2,4)} = \left(\frac{16}{36}, \frac{4}{12}\right)^T, \quad \mathbf{a}_{(0,1,3,3,4)} = \left(\frac{9}{22}\right),$$

$$\mathbf{a}_{(0,1,0,0,3)} = \mathbf{a}_{(0,1,0,0,6)} = \mathbf{a}_{(0,1,0,2,2)} = \mathbf{a}_{(0,1,0,3,3)} = \mathbf{a}_{(0,1,1,0,4)}$$
$$= \mathbf{a}_{(0,1,1,1,6)} = \mathbf{a}_{(0,1,2,0,5)} = \mathbf{a}_{(0,1,2,2,6)} = (1,1)^T,$$

$$\mathbf{a}_{(0,1,3,0,6)} = (1).$$

Note that nine of the aggregation vectors are equal to $\mathbf{e}$ and need not be stored. All of these nine vectors except $\mathbf{a}_{(0,1,1,0,4)}$ are equal to $\mathbf{e}$ since $\mathbf{a}_{(0,0,p,w,k)} = \mathbf{e}$ and submodel 1's transition submatrices $Q_k^{(1)}(\mathcal{R}_p^{(1)}, \mathcal{R}_w^{(1)})$ for the respective transitions k are all diagonal matrices with 1's along their diagonal. For $\mathbf{a}_{(0,1,1,0,4)}$, the situation is slightly different. The submatrix used in the computation of $\mathbf{a}_{(0,1,1,0,4)}$ is given by $Q_4^{(1)}(\mathcal{R}_1^{(1)}, \mathcal{R}_0^{(1)}) = (0,1)$. Because $\mathbf{a}_{(0,0,1,0,4)} = \mathbf{e}$ and submatrix $Q_4^{(1)}(\mathcal{R}_1^{(1)}, \mathcal{R}_0^{(1)})$ is a row vector having a single nonzero with value 1, $\mathbf{a}_{(0,1,1,0,4)}$ also turns out to be equal to $\mathbf{e}$.

Using the aggregation vectors, the ten nonzero blocks of the aggregated generator matrix $\tilde{Q}_{(0,1)}$ are expressed as

$$\tilde{Q}_{(0,1)}(0,0) = \sum_{k\in\{1,3,4,6\}} \alpha_k \operatorname{diag}(\mathbf{a}_{(0,1,0,0,k)}) \bigotimes_{h=2}^{3} Q_k^{(h)}(\mathcal{R}_0^{(h)}, \mathcal{R}_0^{(h)})$$
$$-\sum_{w=0}^{3} \sum_{k\in\mathcal{K}_{0,w}} \alpha_k \operatorname{diag}(\mathbf{a}_{(0,1,0,w,k)}) \bigotimes_{h=2}^{3} \operatorname{diag}(Q_k^{(h)}(\mathcal{R}_0^{(h)}, \mathcal{R}_w^{(h)})\mathbf{e}),$$

$$\tilde{Q}_{(0,1)}(0,1) = \alpha_1 \operatorname{diag}(\mathbf{a}_{(0,1,0,1,1)}) \bigotimes_{h=2}^{3} Q_1^{(h)}(\mathcal{R}_0^{(h)}, \mathcal{R}_1^{(h)}),$$

$$\tilde{Q}_{(0,1)}(0,2) = \alpha_2 \operatorname{diag}(\mathbf{a}_{(0,1,0,2,2)}) \bigotimes_{h=2}^{3} Q_2^{(h)}(\mathcal{R}_0^{(h)}, \mathcal{R}_2^{(h)}),$$

$$\tilde{Q}_{(0,1)}(0,3) = \alpha_3 \operatorname{diag}(\mathbf{a}_{(0,1,0,3,3)}) \bigotimes_{h=2}^{3} Q_3^{(h)}(\mathcal{R}_0^{(h)}, \mathcal{R}_3^{(h)}),$$

$$\tilde{Q}_{(0,1)}(1,0) = \alpha_4 \operatorname{diag}(\mathbf{a}_{(0,1,1,0,4)}) \bigotimes_{h=2}^{3} Q_4^{(h)}(\mathcal{R}_1^{(h)}, \mathcal{R}_0^{(h)}),$$

$$\tilde{Q}_{(0,1)}(1,1) = \alpha_6 \operatorname{diag}(\mathbf{a}_{(0,1,1,1,6)}) \bigotimes_{h=2}^{3} Q_6^{(h)}(\mathcal{R}_1^{(h)}, \mathcal{R}_1^{(h)}),$$
$$-\sum_{w=0}^{1} \sum_{k\in\mathcal{K}_{1,w}} \alpha_k \operatorname{diag}(\mathbf{a}_{(0,1,1,w,k)}) \bigotimes_{h=2}^{3} \operatorname{diag}(Q_k^{(h)}(\mathcal{R}_1^{(h)}, \mathcal{R}_w^{(h)})\mathbf{e}),$$

$$\tilde{Q}_{(0,1)}(2,0) = \alpha_5 \ \mathrm{diag}(\mathbf{a}_{(0,1,2,0,5)}) \bigotimes_{h=2}^{3} Q_5^{(h)}(\mathcal{R}_2^{(h)}, \mathcal{R}_0^{(h)}) \ ,$$

$$\tilde{Q}_{(0,1)}(2,2) = \sum_{k\in\{4,6\}} \alpha_k \ \mathrm{diag}(\mathbf{a}_{(0,1,2,2,k)}) \bigotimes_{h=2}^{3} Q_k^{(h)}(\mathcal{R}_2^{(h)}, \mathcal{R}_2^{(h)})$$
$$- \sum_{w\in\{0,2\}} \sum_{k\in\mathcal{K}_{2,w}} \alpha_k \ \mathrm{diag}(\mathbf{a}_{(0,1,2,w,k)}) \bigotimes_{h=2}^{3} \mathrm{diag}(Q_k^{(h)}(\mathcal{R}_2^{(h)}, \mathcal{R}_w^{(h)})\mathbf{e}) \ ,$$

$$\tilde{Q}_{(0,1)}(3,0) = \alpha_6 \ \mathrm{diag}(\mathbf{a}_{(0,1,3,0,6)}) \bigotimes_{h=2}^{3} Q_6^{(h)}(\mathcal{R}_3^{(h)}, \mathcal{R}_0^{(h)}) \ ,$$

$$\tilde{Q}_{(0,1)}(3,3) = \alpha_4 \ \mathrm{diag}(\mathbf{a}_{(0,1,3,3,4)}) \bigotimes_{h=2}^{3} Q_4^{(h)}(\mathcal{R}_3^{(h)}, \mathcal{R}_3^{(h)})$$
$$- \sum_{w\in\{0,3\}} \sum_{k\in\mathcal{K}_{3,w}} \alpha_k \ \mathrm{diag}(\mathbf{a}_{(0,1,3,w,k)}) \bigotimes_{h=2}^{3} \mathrm{diag}(Q_k^{(h)}(\mathcal{R}_3^{(h)}, \mathcal{R}_w^{(h)})\mathbf{e}) \ .$$

Regarding the five aggregation vectors that are computed to be different than $\mathbf{e}$, $\mathbf{a}_{(0,1,0,0,1)}$, $\mathbf{a}_{(0,1,0,0,4)}$, $\mathbf{a}_{(0,1,2,2,4)}$, and $\mathbf{a}_{(0,1,3,3,4)}$ are used in aggregating diagonal blocks of $\tilde{Q}_{(0,0)}$, and their contributions are to the diagonals of diagonal blocks. The contributions of $\mathbf{a}_{(0,1,0,0,1)}$ and $\mathbf{a}_{(0,1,0,0,4)}$ in the first summation of $\tilde{Q}_{0,1}(0,0)$ are to the diagonal since $Q_1^{(2)}(\mathcal{R}_0^{(2)}, \mathcal{R}_0^{(2)}) = Q_4^{(2)}(\mathcal{R}_0^{(2)}, \mathcal{R}_0^{(2)}) = I_1$ and $Q_1^{(3)}(\mathcal{R}_0^{(3)}, \mathcal{R}_0^{(3)}) = Q_4^{(3)}(\mathcal{R}_0^{(3)}, \mathcal{R}_0^{(3)}) = I_2$. But their effects are canceled by the second negated summation of $\tilde{Q}_{0,1}(0,0)$ simply because $\mathrm{diag}(Q_1^{(2)}(\mathcal{R}_0^{(2)}, \mathcal{R}_0^{(2)})\mathbf{e}) = \mathrm{diag}(Q_4^{(2)}(\mathcal{R}_0^{(2)}, \mathcal{R}_0^{(2)})\mathbf{e}) = I_1$ and $\mathrm{diag}(Q_1^{(3)}(\mathcal{R}_0^{(3)}, \mathcal{R}_0^{(3)})\mathbf{e}) = \mathrm{diag}(Q_4^{(3)}(\mathcal{R}_0^{(3)}, \mathcal{R}_0^{(3)})\mathbf{e}) = I_2$. This is also the case for $\mathbf{a}_{(0,1,2,2,4)}$ and $\mathbf{a}_{(0,1,3,3,4)}$. Hence, we may very well set these four aggregation vectors to $\mathbf{e}$ as suggested before. Therefore, from $\tilde{Q}_{(0,1)} = P_{\mathbf{x}_{(0,0)}} \tilde{Q}_{(0,0)} R_{(0)}$, we implicitly have

$$\tilde{Q}_{(0,1)} = \begin{array}{c} \\ 0\ 0 \\ 0\ 1 \\ 0\ 0 \\ 0\ 1 \\ 1\ 0 \\ 1\ 1 \\ 0\ 2 \end{array} \begin{array}{c} \begin{array}{cccccccc} 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 2 \end{array} \\ \left(\begin{array}{cc|cc|cc|c} -\frac{280}{33} & 6 & \frac{16}{33} & & 2 & & \\ 6 & -\frac{448}{39} & & \frac{19}{39} & & 2 & 3 \\ \hline 2 & & -2 & & & & \\ & 2 & 6 & -8 & & & \\ \hline 4 & & & & -4 & & \\ & 4 & & & 6 & -10 & \\ \hline & 6 & & & & & -6 \end{array}\right) \end{array} .$$

In the next step, similar operations will be carried out at level 1 unless the aggregated generator matrix is solved exactly, upon which backtracking from recursion starts for iteration it.

The ML method we discussed follows a *V–cycle* [205, 206] at each iteration. That is, starting from the finest level, at each step, it smooths the current approximation and moves to a coarser level by aggregation until it reaches a level at which the aggregated generator matrix 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 smooths it. Furthermore, the submodel state spaces, $\mathcal{S}^{(h)}$, are aggregated according to the *fixed* order $h = 1, \ldots, H$. However, to the contrary of the ML method for sparse MCs in [187], the definition of the aggregated state spaces follows naturally from the Kronecker representation in (2.10), and the aggregated generator matrices can also be represented using Kronecker products as shown in (6.23).

In [59], a sophisticated class of ML methods are given. The methods therein are capable of using JOR and SOR as smoothers, performing the *W–* and *F–cycles* inspired by *multigrid* [39, 276, 324] and aggregating submodel state spaces according to *circular* and *adaptive* orders. Here, a *cycle* may be viewed as the operations corresponding to an outer ML iteration denoted by *it* in (6.16), and W–cycle refers to invoking at each level two recursive calls to the next coarser level, whereas an F–cycle at a level can be viewed as a recursive call to a W–cycle followed by a recursive call to a V–cycle at the next coarser level. In the circular order of aggregation, at the beginning of each ML cycle at the finest level, a circular shift of submodel indices is performed to achieve fairness in aggregating submodel state spaces. Hence, every H cycles each submodel will have received the opportunity to get aggregated first. On the other hand, in the adaptive order of aggregation, submodel indices are sorted according to the residual norms restricted to the corresponding submodel state space at the end of the ML cycle, and aggregation of submodels in this sorted order in the next cycle is performed. This ensures that submodels which have smaller residual norms are aggregated earlier at finer levels, since small residual norms are expected to be indicative of good numerical results in those submodels. ML methods discussed in this section are coded within the `NSolve` package of the APNN toolbox [7, 22]. Numerical experiments in [59, 61] prove these 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 [59] are discussed in [61]. An almost positive row or column in the iteration matrices of the smoother with $\eta_{(it,l)}$ pre- and $\nu_{(it,l)}$ post-smoothings at each level l across iterations it is shown to facilitate convergence locally, and SOR is to be recommended among the three smoothers considered. An error propagation formula for ML methods in which the number of pre-smoothings is set to zero is introduced in [272]. Therein are examples indicating that a two-level method may be converging while the corresponding three-level method is not and vice versa. One important question to be answered is whether local convergence implies global convergence (even at two levels). Furthermore, it

is not clear how the behavior of the ML method 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 within the ML method, since the aggregated generator matrix at each level changes from iteration to iteration and the LU factorization of the diagonal blocks may be too time consuming to offset.

In [173] an efficient algorithm that finds an NCD [97, 123, 241, 305] partitioning of $\mathcal{S}$ 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 [304] (cf. [225, 228, 271]), 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.

Distributed implementation of block iterative methods coupled with some number of aggregation–disaggregation steps for HMMs is investigated on a cluster of workstations in [67]. Therein, especially asynchronous computation models that observe coarse-grained parallelism (meaning large chunks of computations interleaved with small amount of communications) based on the nested block partitioning of Q are recommended at lower level numbers. Along a different line, the performance of a prototype parallel version of PEPS on a cluster architecture is modeled using SANs and analyzed theoretically with a sequential version of PEPS [14]. The study in [288] concentrates on the parallelization of the vector operations of initialization, reciprocation, aggregation, disaggregation, scaling, addition, multiplication and the matrix operations of vector–matrix multiplication and aggregation in symbolic ML [287] with V–cycle, fixed order of submodel aggregation, and JOR smoother under the *multiterminal binary decision diagram* (MTBDD) data structure using multiple threads. Clearly, there are issues that need to be addressed further such as the level at which parallelism is employed, resolving data collision due to multiple threads accessing data that contributes to the same intermediate result and load balancing across threads. The results however are quite encouraging and indicate that there is ample opportunity for parallelism in the implementation of iterative methods based on the block partitioning of Q and the sum of Kronecker products forming each nonzero block.

6.4 Decompositional Methods

Among the iterative methods discussed in the previous sections, ML methods perform better on a larger number of problems in the literature [59, 61]. However, there are certain classes of problems for which other methods could be preferred. One such method we present in this section is iterative and based on *decomposing* a model into its submodels, analyzing the submodels individually for their steady-state, and putting back the individual solutions

together using disaggregation in a correction step [15]. This method is able to compute the steady-state solution exactly up to computer precision to a model having weakly interacting submodels in a relatively small number of iterations and with modest memory requirements.

Consider the separation of Q_O in (6.2) into two terms as in

$$Q_O = Q_{local} + Q_{synch} \ ,$$

where Q_{local} and Q_{synch} correspond to those parts of Q_O associated with local and synchronizing transitions, respectively. Without loss of generality, we adopt the enumeration of the K terms as in Example 1 and let the first H represent local transitions, the hth being associated with submodel h for $h = 1, \ldots, H$. The remaining $(K - H)$ terms correspond to synchronizing transitions. Hence,

$$Q_{local} = \sum_{k=1}^{H} \bigotimes_{h=1}^{H} Q_k^{(h)} \quad \text{and} \quad Q_{synch} = \sum_{k=H+1}^{K} \bigotimes_{h=1}^{H} Q_k^{(h)} \ .$$

Recall that this enumeration necessarily implies $Q_k^{(h)} = I_{n_h}$ for $k \neq h$ and $h = 1, \ldots, H$ due to the definition of local transitions. In Example 1, we have $\mathcal{S} = \mathcal{R}$; hence, there is a single reachable state space partition for which $|\mathcal{R}| = \prod_{h=1}^{H} |\mathcal{S}^{(h)}| = \prod_{h=1}^{H} |\mathcal{R}^{(h)}| = \prod_{h=1}^{H} n_h = n$. Furthermore, we observe that the irreducibility of Q does not imply the irreducibility of the local transition rate matrices $Q_h^{(h)} \in \mathbb{R}_{\geqslant 0}^{n_h \times n_h}$ for $h = 1, \ldots, H$.

Now, let $Q = U - L$ be the forward GS splitting of the generator matrix in Kronecker form, where $U = Q_D + Q_{U(H)}$ corresponds to its upper-triangular part and $L = Q_{L(H)}$ contains its negated strictly lower-triangular part as in (6.8). Furthermore, let the aggregation operator $R^{(h)} \in \mathbb{R}_{\geqslant 0}^{n \times n_h}$ (cf. (6.18)) for $h = 1, \ldots, H$ be associated with the mapping $f_{(h)} : \mathcal{R} \mapsto \mathcal{R}^{(h)}$ and have its $(\mathbf{i}, i_h)$th entry be given by

$$r^{(h)}(\mathbf{i}, i_h) = \begin{cases} 1 & \text{if } f_{(h)}(\mathbf{i}) = i_h \\ 0 & \text{otherwise} \end{cases} \quad \text{for } \mathbf{i} \in \mathcal{R} \text{ and } i_h \in \mathcal{R}^{(h)} \ .$$

Observe that the mapping $f_{(h)}$ represents the aggregation of all dimensions except the hth. In Kronecker form,

$$R^{(h)} = \left(\bigotimes_{l=1}^{h-1} I_{n_l} \mathbf{e} \right) \otimes I_{n_h} \otimes \left(\bigotimes_{l=h+1}^{H} I_{n_l} \mathbf{e} \right) \quad \text{for } h = 1, \ldots, H \ . \tag{6.24}$$

On the other hand, let the disaggregation operator $S_{\boldsymbol{\pi}_{(it)}}^{(h)} \in \mathbb{R}_{\geqslant 0}^{n_h \times n}$ (cf. (6.19)) for $h = 1, \ldots, H$ be associated with the mapping $f_{(h)}$ and have its $(i_h, \mathbf{i})$th entry be given by

$$s^{(h)}_{\boldsymbol{\pi}_{(it)}}(i_h, \mathbf{i}) = \begin{cases} \frac{\boldsymbol{\pi}_{(it)}(\mathbf{i})}{\boldsymbol{\pi}^{(h)}_{(it)}(i_h)} & \text{if } f_{(h)}(\mathbf{i}) = i_h \\ 0 & \text{otherwise} \end{cases} \quad \text{for } \mathbf{i} \in \mathcal{R} \text{ and } i_h \in \mathcal{R}^{(h)} ,$$

where

$$\boldsymbol{\pi}^{(h)}_{(it)} := \boldsymbol{\pi}_{(it)} R^{(h)}.$$

Then the decompositional iterative method can be stated [15] for a user-specified `stop_tol` as in Algorithm 8.

ALGORITHM 8. *Decompositonal iterative method with GS correction step.*

$it := 0$; $\mathbf{y}_{(it)} := \mathbf{0}$; $\boldsymbol{\pi}_{(it)} := \mathbf{e}^T/n$;
Repeat
  For $h := 1$ to H,
    If $Q^{(h)}_h$ is irreducible, solve $\boldsymbol{\pi}^{(h)}_{(it+1)} Q^{(h)}_h = \mathbf{v}^{(k)}(\boldsymbol{\pi}_{(it)})$,
      where $\mathbf{v}^{(k)}(\boldsymbol{\pi}_{(it)}) := -\boldsymbol{\pi}_{(it)} Q_{synch} R^{(h)}$,
    Else solve $\boldsymbol{\pi}^{(h)}_{(it+1)} Q^{(h)}(\boldsymbol{\pi}_{(it)}) = \mathbf{0}$,
      where $Q^{(h)}(\boldsymbol{\pi}_{(it)}) := Q^{(h)}_h + S^{(h)}_{\boldsymbol{\pi}_{(it)}} Q_{synch} R^{(h)}$,
    subject to $\boldsymbol{\pi}^{(h)}_{(it+1)} \mathbf{e} = 1$;
  Solve $\mathbf{y}_{(it+1)} U = \mathbf{y}_{(it)} L + \left(\bigotimes_{h=1}^{H} \boldsymbol{\pi}^{(h)}_{(it+1)} \right) Q$;
  $\boldsymbol{\pi}_{(it+1)} := \left(\bigotimes_{h=1}^{H} \boldsymbol{\pi}^{(h)}_{(it+1)} \right) - \mathbf{y}_{(it+1)}$ subject to $\boldsymbol{\pi}_{(it+1)} \mathbf{e} = 1$;
  $it := it + 1$;
Until $\|\boldsymbol{\pi}_{(it)} Q\|_\infty <$ `stop_tol`.

The algorithm starts by initializing the correction vector, $\mathbf{y}_{(it)}$, to zero and the solution vector, $\boldsymbol{\pi}_{(it)}$, to the uniform distribution, respectively. Then each system of local equations is solved subject to a normalization condition. If $Q^{(h)}_h$ is irreducible, then a unique new local solution vector $\boldsymbol{\pi}^{(h)}_{(it+1)} \in \mathbb{R}^{1 \times n_h}_{>0}$ can be computed. This is so, because each system to be solved has a zero sum right-hand side vector, $\mathbf{v}^{(h)}(\boldsymbol{\pi}_{(it)})$, (i.e., $\mathbf{v}^{(h)}(\boldsymbol{\pi}_{(it)})\mathbf{e} = 0$) due to the particular way in which synchronizing transition rate matrices, $Q^{(h)}_k \in \mathbb{R}^{n_h \times n_h}_{\geqslant 0}$ for $k = H+1, \ldots, K$, are specified. On the other hand, when $Q^{(h)}_h$ is reducible, we consider a homogeneous system in which the aggregated matrix $Q^{(h)}(\boldsymbol{\pi}_{(it)})$ is used. The aggregated matrix is irreducible if Q is irreducible and $\boldsymbol{\pi}_{(it)} \in \mathbb{R}^{1 \times n}_{>0}$. Hence, the existence of a unique $\boldsymbol{\pi}^{(h)}_{(it+1)} \in \mathbb{R}^{1 \times n_h}_{>0}$ is also guaranteed in this case. Since $Q^{(h)}_k$ for $k = H+1, \ldots, K$ are in general very sparse, the enumeration process associated with the nonzeros in Q_{synch} to form $\mathbf{v}^{(k)}(\boldsymbol{\pi}_{(it)})$, or $Q^{(h)}(\boldsymbol{\pi}_{(it)})$, can be handled systematically. Note that there are differences from a computational point of view between the two alternative solution steps. In the former case, $Q^{(h)}_h$ is constant and already available in row sparse format; the right-hand side vector is dependent on

$\boldsymbol{\pi}_{(it)}$. In the latter case, $Q^{(h)}(\boldsymbol{\pi}_{(it)})$ needs to be reconstructed at each iteration, and it is the coefficient matrix that is dependent on $\boldsymbol{\pi}_{(it)}$. Hence, the two approaches for obtaining $\boldsymbol{\pi}^{(h)}_{(it+1)}$ are not equivalent except at steady-state (i.e., $\boldsymbol{\pi}_{(it)} = \boldsymbol{\pi}_{(it+1)} = \boldsymbol{\pi}$).

In the next step, the new correction vector, $\mathbf{y}_{(it+1)}$, is obtained by solving an upper-triangular system in Kronecker form. Since Q is assumed to be irreducible, $\mathbf{y}_{(it+1)}$ is computed through a GS relaxation on Q with a zero sum but nonzero right-hand side.

The last step subtracts $\mathbf{y}_{(it+1)}$ from the Kronecker product of $\boldsymbol{\pi}^{(h)}_{(it+1)}$ for $h = 1, \ldots, H$ to form the new solution vector, $\boldsymbol{\pi}_{(it+1)}$, and then the iteration number is incremented. These steps are repeated until the residual infinity norm becomes smaller than `stop_tol`.

Algorithm 8 is coded within the `NSolve` package of the APNN toolbox [7, 22] and numerical experiments are carried out [15] on larger and slightly different versions of Example 1. The decompositional method is compared with point iterative methods based on splittings, BGS, projection methods, BGS preconditioned projection methods, and ML with one pre- and one post-smoothing using GS, W–cycle, and circular order of aggregating submodels in each cycle. The solvers are compared in terms of number of iterations to converge to `stop_tol` $:= 10^{-8}$ on the infinity norm of the residual, elapsed CPU time, and amount of allocated main memory. The diagonal blocks associated with the BGS solver and the BGS preconditioner for projection methods at a suitable level number are LU factorized [58] using `colamd` ordering [94, 95]. The number of nonzeros generated during the LU factorization of the diagonal blocks is accounted for in the memory consumed by solvers utilizing BGS.

It is observed that convergence becomes very fast for the decompositional solver when the synchronizing transition rates are small since the submodels in that case are nearly independent and the Kronecker product of the local solutions yields a very good approximation to the solution early in the iteration. The solver corresponding to Algorithm 8 requires modest memory and a small number of iterations for relatively fast convergence to the solution. The second best solver is ML, which trails in all three areas. It improves only slightly as the synchronizing transition rates become smaller. Projection methods do not benefit from BGS preconditioning. The performances of the point iterative methods based on splittings and BGS are not affected by a change in the rates of synchronizing transitions. BGS performs very poorly due to the large time per iteration. BGS and BGS preconditioned projection methods require considerably more memory than the other methods, because of the need to store factors of diagonal blocks and, in the latter case, also a larger number of vectors. Memorywise, the decompositional solver requires about 1.5 times that of point iterative methods based on splittings, but less than ML, and therefore can be considered to be memory efficient.

The scalability of the decompositional solver is investigated for increasing number of submodels when the synchronizing transition rates are relatively small compared to those in local transition rate matrices. It is observed that the number of iterations to converge decreases as the number of submodels increases. This is due to the decrease in the throughputs of synchronizing transitions for a larger number of submodels (because the steady-state probabilities of states in which synchronizing transitions can take place become smaller), leading to more independent submodels. This is different from the behavior of the ML method, which takes more or less the same number of iterations to converge as the number of submodels increases. The scalability of the decompositional solver is also investigated for increasing number of synchronizing transitions when the number of submodels is kept constant and the rates of synchronizing transitions are relatively small compared to those in the local transition rate matrices. As expected, the results indicate that the time the decompositional solver takes to converge is affected linearly by an increase in the number of synchronizing transitions.

There are also iterative methods based on *polyhedra* theory [86] and disaggregation, such as the one in [55] for SANs which provides satisfactory lower and upper *bounds* on the solution only if the interactions among submodels are weak or the rates of synchronizing transitions are more or less independent of the states of submodels. Another class of iterative methods are those that are *approximative*. For instance, the method in [56] for superposed GSPNs operates at a fine level only on states having higher steady-state probabilities; the remaining states are aggregated and treated at a coarse level. The steady-state vector can be stored with significant savings due to its compact representation as a Kronecker product of the aggregated submodel steady-state vectors. An approximative class of iterative methods are also presented in [101]. The approximative methods therein are geared toward *closed* networks of first-come first-served (FCFS) queues with PH service distributions and arbitrary buffer sizes when a few digits of accuracy in the computed solution are sufficient for analysis purposes [104]. The analysis of closed queueing networks with PH service distributions and arbitrary buffer sizes is challenging due to the fact that the corresponding state spaces grow exponentially with numbers of customers, queues, and phases in the service distribution of each queue. Now, we discuss these approximative iterative methods for closed queueing networks, which are also based on decomposition. More information regarding this work is available in [104, 239].

Queueing networks have been used in the literature to model and analyze a variety of systems involving customers, packets, or jobs waiting to get service [180, 181, 306]. The work in [104] concentrates on a relatively large class of problems which do not possess analytical solutions [153]. Closedness implies that the number of customers circulating in the queueing network remains constant; there are no arrivals to the network from the outside, there are no departures to the outside, and the number of customers inside the network neither increases nor decreases as a result of the queueing discipline and the

service process. A customer departs from a queue after getting service and joins a(nother) queue, possibly the same one it departed from. If a queueing network is not closed, it is said to be *open*. Regarding service distributions, *hypoexponential*, *hyperexponential*, *Coxian*, and *Erlang* are all phase-type and have rational Laplace transforms. Furthermore, the exponential distribution is a special case of the Erlang distribution, which is yet a special case of the hypoexponential distribution. Interestingly, it is proved that Erlang is the most suitable phase approximation for the *deterministic* distribution [6]. This is taken advantage of when modeling a robotic tape library [119] and a multiprocessor system [285] using SANs. For practical purposes, a five- to ten-phase Erlang is considered sufficient for approximating a deterministic distribution. The use of PH distributions in SANs is further investigated in [284].

In [104], two approximative decompositional iterative methods are addressed, the first of which appears in [235] and the second one in [329]. Each decomposes the network into subnetworks. They differ in the way the decompositions are obtained, and the solutions to subnetworks are put together. These approximative methods require the modeling of subnetworks whose product state space sizes are larger than their reachable state space sizes and are shown that they can be implemented using Kronecker products.

In its setup phase, the first method in [235] partitions the closed queueing network into subnetworks. In doing this, it classifies each queue as finite or infinite buffer. Finite buffer queues are those that have positive blocking probabilities for a given total number of customers circulating in the closed queueing network. The method places queues feeding a finite buffer queue in the same subnetwork with the finite buffer queue. In this way, the method aims at achieving a decomposition in which transition probabilities between subnetworks are independent of the states of the subnetworks. Thus, each subnetwork can be considered as a service station with state-dependent exponential service rate for which the parameters of the equivalent server are obtained by analyzing the subnetwork in isolation as an open queueing network assuming it has state-dependent arrivals with exponentially distributed interarrival times. This is done by modeling the open queueing network as a closed queueing network, which consists of the subnetwork's queues and a slack queue as in Figure 6.1. The slack queue is assumed to have a finite buffer of size equal to the total number of customers circulating in the closed queueing network formed of multiple subnetworks. The state-dependent throughputs are the state-dependent service rates since the slack queue is practically infinite. Each closed queueing network obtained as such can be modeled by defining the queues in the subnetwork and the slack queue, and then constructing a block matrix which represents the interactions among the queues in the closed queueing network using Kronecker products as in [43]. The approximate results are obtained via *fixed-point iteration*, which requires throughputs of subnetworks to be computed. For this purpose, an analytical method, the convolution algorithm (CA) [160] is used. We remark that

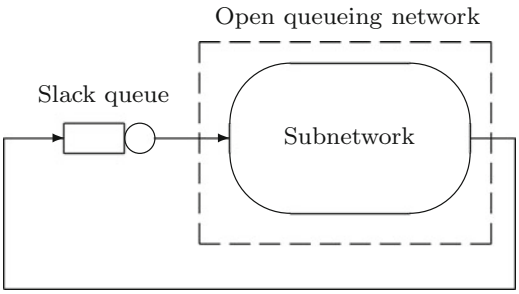

Fig. 6.1 Open queueing network corresponding to subnetwork modeled as closed queueing network with slack queue having finite buffer.

a fixed-point iteration can be perceived, for instance, as an iteration of the form in block iterative methods, for which the solution, π, that is sought is the *fixed-point* of the system of equations, $\pi = \pi T$, to be solved [305].

On the other hand, the second decompositional method in [329] partitions the closed queueing network into individual queues and approximates the service distribution of each queue by a state-dependent exponential service distribution. Thus, the method transforms the closed queueing network into another closed queueing network with state-dependent exponential service distributions. The decomposition in this approach is maximal, meaning each queue is placed in a separate subnetwork. Again a slack queue with an infinite buffer and a state-dependent exponential service distribution is used to model state-dependent arrivals with exponentially distributed interarrival times to each queue having a PH service distribution. After this approximation, the method sets the state-dependent service rate of the slack queue to some initial value and then employs a fixed-point iteration on the decomposed network to compute the throughputs of all queues. Again, initialization of the state-dependent service rates of slack queues, which are their state-dependent throughputs, can be done using CA.

Implementations of the two approximative decompositional methods (respectively, named M and YB hereafter) are available in Matlab [240] together with implementations of CA, a *mean value analysis algorithm for blocking closed queueing networks* (MVABLO) [5], point iterative methods based on splittings, and ML with fixed and circular orders of aggregation in V–, F–, and W–cycles. Experiments are performed to compare the methods for their accuracy and efficiency on various models for analyzing utilizations and mean lengths of queues. ML and GS methods assume `stop_tol` $:= 10^{-15}$ on the residual 1-norm. ML uses GS as the smoother and performs one pre- and one post-smoothing at each level. A `stop_tol` $:= 10^{-4}$ is used on the approximate error of utilizations and mean lengths of queues for M and YB. The subnetworks resulting from decomposition in these methods are solved with ML. When computing the steady-state vector of the coarsest generator matrix in M and the steady-state vector of the state-dependent closed

queueing network with exponential service distributions in YB, if the order of the matrix is less than 500, GE, otherwise BICGSTAB with ILU preconditioning and a drop tolerance of 10^{-5} [123] is employed. Results obtained by approximative methods are compared with results of ML and , relative errors are provided using the 1-norm. Note that relative errors are indicative of numbers of correct decimal digits in the results. That is, an approximate result with a relative error in the order of 10^{-z} implies z correct decimal digits.

CA and MVABLO produce acceptable results for problems with balanced service requirements and relatively small number of blocking queues. On the other hand, M and YB provide relatively more accurate results for all problems and yield results with at least 2 digits of accuracy for unbalanced service requirements. Also, unlike the results obtained with CA and MVABLO, an increase in the number of blocking queues has almost no effect in the results obtained with M and YB. Therefore, M and YB emerge as more accurate methods than CA and MVABLO for problems with unbalanced service requirements and relatively large number of blocking queues. When accuracies of M and YB are compared, especially in problems with unbalanced service requirements and relatively large number of blocking queues, M can produce at least half a digit more accurate results for utilizations than YB.

When efficiencies of M and YB are compared, it can be seen that the number of flops performed by YB to compute arrival rates of queues mostly depends on the number of flops performed for obtaining the solution of the state-dependent closed queueing network with exponential service distributions generated at each fixed-point iteration. Therefore, for problems which require a relatively small number of flops for the solution of this queueing network, YB executes less flops than M. Also, for problems which result in subnetworks with a relatively large number of queues for M, YB may end up performing less flops than M through its fixed-point approximation process. Consequently, efficiencies of M and YB depend heavily on the particular problem. Nevertheless, the average number of fixed-point iterations performed by M and YB over all problems considered are 4 and 5, respectively. When ML and GS are compared, we see that ML achieves convergence within 100 iterations in all problems. On the other hand, GS does not converge within a reasonable number of iterations or time in some of the problems. Clearly, the number of iterations determines the number of flops executed by the methods, and ML performs less flops than GS in almost all problems. Even though GS takes less space in memory than ML, in most of the problems, ML requires less memory than the sparse representation of the generator matrix underlying the closed queueing network with PH service distributions and arbitrary buffer sizes, thereby being capable of solving variants of the problems with relatively large numbers of customers. Since M and YB are based on decomposition, the space requirements of M and YB are smaller than those of ML and GS for relatively large problems. Indeed, it is verified that the usage of ML in M and YB introduces another dimension of scalability to the space requirements of the two methods.

6.5 Matrix Analytic Methods

Continuous-time LDQBDs are CTMCs having generator matrices that can be symmetrically permuted to the block tridiagonal form

$$Q = \begin{pmatrix} Q(0,0) & Q(0,1) & & & & \\ Q(1,0) & Q(1,1) & Q(1,2) & & & \\ & \ddots & \ddots & \ddots & & \\ & & Q(l,l-1) & Q(l,l) & Q(l,l+1) & \\ & & & \ddots & \ddots & \ddots \end{pmatrix}. \tag{6.25}$$

As opposed to QBDs [4, 33, 213], the dependency on level number, l, in (6.25) manifests itself with two indices rather than one in each nonzero block for $l \in \mathbb{Z}_{\geqslant 0}$. It is the first index which corresponds to level number. Nevertheless, nonzero blocks can be dependent on level number in two different ways. It is either the nonzero values or the dimensions of a nonzero block (or both) that depend on level number. In this respect, LDQBDs generalize QBDs with nonhomogeneous transition rates and rectangular subdiagonal/superdiagonal nonzero blocks.

Assuming that the subset of states corresponding to level l is denoted by $\mathcal{R}_l$, the nonzero blocks at level l are given by

$$Q(l,l-1) \in \mathbb{R}_{\geqslant 0}^{|\mathcal{R}_l| \times |\mathcal{R}_{l-1}|}, \quad Q(l,l) \in \mathbb{R}^{|\mathcal{R}_l| \times |\mathcal{R}_l|}, \quad Q(l,l+1) \in \mathbb{R}_{\geqslant 0}^{|\mathcal{R}_l| \times |\mathcal{R}_{l+1}|}.$$

Negative entries appear only along the diagonal of $Q(l,l)$. There are a countably infinite number of levels, and transitions from level l are either to states within itself or to states in the adjacent levels $(l-1)$ and $(l+1)$. Level 0 is an exception since it constitutes the boundary level and has two nonzero blocks. Clearly, the ordering of states within a level is fixed only up to a permutation.

Assuming that steady-state exists, thereof the probability distribution vector may be written in a piecemeal manner as

$$\boldsymbol{\pi} = (\boldsymbol{\pi}(\mathcal{R}_0), \boldsymbol{\pi}(\mathcal{R}_1), \ldots),$$

and its subvector at level $(l+1)$ can be obtained from

$$\boldsymbol{\pi}(\mathcal{R}_{l+1}) = \boldsymbol{\pi}(\mathcal{R}_l) R_l \tag{6.26}$$

once the *matrix of conditional expected sojourn times* at level l

$$R_l = Q(l,l+1)(-Q(l+1,l+1) - R_{l+1}Q(l+2,l+1))^{-1} \tag{6.27}$$

is available for $l \in \mathbb{Z}_{\geqslant 0}$ [40]. In (6.27), $R_l(\mathbf{i},\mathbf{j})$ records the expected sojourn time in state $\mathbf{j} \in \mathcal{R}_{l+1}$ per unit sojourn in state $\mathbf{i} \in \mathcal{R}_l$ before returning to level l, given that the process started in state $\mathbf{i}$ [274]. We remark that

$$R_l \in \mathbb{R}_{\geqslant 0}^{|\mathcal{R}_l| \times |\mathcal{R}_{l+1}|} \quad \text{for } l \in \mathbb{Z}_{\geqslant 0}$$

is nonnegative and rectangular. The recurrence in (6.26) requires $\boldsymbol{\pi}(\mathcal{R}_0)$ to be determined first. This can be done from the set of boundary equations

$$\boldsymbol{\pi}(\mathcal{R}_0)Q(0,0) + \boldsymbol{\pi}(\mathcal{R}_1)Q(1,0) = \mathbf{0}$$

corresponding to the first column of blocks using $\boldsymbol{\pi}(\mathcal{R}_1) = \boldsymbol{\pi}(\mathcal{R}_0)R_0$ from s(6.27). Hence, we conclude that $\boldsymbol{\pi}(\mathcal{R}_0)$ should be the positive left eigenvector of $Q(0,0) + R_0Q(1,0)$ corresponding to the eigenvalue 0 [242]. Eventually, $\boldsymbol{\pi}$ should be normalized so that $\boldsymbol{\pi}\mathbf{e} = 1$.

In Section 4.4, we have discussed how the countable infiniteness of the reachable state space

$$\mathcal{R} = \bigcup_{p=0}^{\infty} \mathcal{R}_p$$

in (4.5) can be handled during steady-state analysis when Q is ergodic using a suitable Lyapunov function [157, 314]. We have also discussed how states are assigned to reachable state space partitions, $\mathcal{R}_p$, and had given a Kronecker representation for each nonzero block. In fact, reachable state space partitions as defined in the metabolite synthesis and call center models correspond to levels in LDQBDs, that is, $\mathcal{R}_p = \mathcal{R}_l$ for $p = l$ and $p, l \in \mathbb{Z}_{\geqslant 0}$, since Q in each of these models for the given partitioning of the reachable state space $\mathcal{R}$ is block tridiagonal. It should be apparent from the values $\mathbf{i} \in \mathcal{R}$ can take that the state space $\mathcal{R}$ is countably infinite and an LDQBD model requires us to truncate it judiciously for analysis purposes.

In many cases, an LDQBD can be shown to be ergodic by checking easy to verify conditions on the *birth-and-death process* (i.e., a one-dimensional CTMC in which transitions from state i to states $i-1$, a death, and $i+1$, a birth, is possible) defined over its levels [268]. However, for computational purposes, we prefer to consider Lyapunov functions, since it is through this approach that lower and higher level numbers (called Low and $High$, respectively) can be computed [103, 107, 120] between which a specified percentage of the steady-state probability mass lies when the LDQBD is ergodic. Once we have proved the finiteness of $\mathcal{C}$ and determined χ (or equivalently, γ for chosen ε) with the help of a suitable Lyapunov function using (4.3), we can compute the pair of level numbers, $(Low, High)$, of the LDQBD such that the states within levels Low to $High$ include all the states in $\mathcal{C}$. In other words, we set

$$Low = \min\{l \in \mathbb{Z}_{\geqslant 0} \mid \mathcal{R}_l \cap \mathcal{C} \neq \varnothing\} \quad \text{and} \quad High = \max\{l \in \mathbb{Z}_{\geqslant 0} \mid \mathcal{R}_l \cap \mathcal{C} \neq \varnothing\}\,,$$

and the finite set

$$\tilde{\mathcal{R}} := \bigcup_{l=Low}^{High} \mathcal{R}_l$$

contains at least $1-\varepsilon$ of the steady-state probability. We remark that $\mathcal{C} \subseteq \tilde{R}$ due to the way in which Low and $High$ are defined, but using a truncated set of states which is a superset of $\mathcal{C}$ can only improve the quality of the bound.

Given the pair of level numbers, $(Low, High)$, the next step is to compute the R_l matrices of conditional expected sojourn times for levels between Low and $High$ using (6.27). This requires us to determine a starting value for R_{High}. Since R_{High} can only be approximated when the state space is truncated, all computed R_l matrices of conditional expected sojourn times between levels Low and $High$ will also be approximate. Clearly, the quality of the approximations improves as the steady-state probability mass concentrated on states within levels Low and $High$ approaches 1 (i.e., as ε approaches 0) [120]. Through a set of experiments in [120], it is shown that setting R_{High} to 0 as suggested in [20] results in almost no loss of accuracy when ε is close to 0 and is the best overall choice.

The matrix analytic computation of the steady-state vector of an LDQBD is given in Algorithm 9 [19]. Its implementation within the `NSolve` package of the APNN toolbox [7, 22] is available at [109]. At step l, the linear system of equations

$$\tilde{R}_l A_l = B_l \quad \text{for } l = High - 1 \text{ down to } Low$$

is solved for the rectangular matrix $\tilde{R}_l \in \mathbb{R}_{\geqslant 0}^{|\mathcal{R}_l| \times |\mathcal{R}_{l+1}|}$ of unknowns, where

$$A_l := Q(l+1, l+1) + \tilde{R}_{l+1} Q(l+2, l+1)$$

is the square coefficient matrix such that $A_l \in \mathbb{R}^{|\mathcal{R}_{l+1}| \times |\mathcal{R}_{l+1}|}$ and

$$B_l := -Q(l, l+1)$$

is the rectangular matrix of multiple right-hand sides with $B_l \in \mathbb{R}^{|\mathcal{R}_l| \times |\mathcal{R}^{l+1}|}$. Clearly, $\tilde{R}_{l+1}$ must be available at step l for the computation to proceed. That R_l and $\pi(\mathcal{R}_l)$ become approximations once R_{High} is set to 0 is indicated by using them with tilde.

The computation of $\tilde{R}_l$ requires the nonzero blocks $Q(l+1, l+1)$ and $Q(l+2, l+1)$ to be obtained, A_l to be formed by multiplying $\tilde{R}_{l+1}$ with $Q(l+2, l+1)$, and then $Q(l+1, l+1)$ to be added to the product. At the end, A_l should be LU factorized and the linear system solved for each right-hand side vector in B_l. The matrix–matrix multiplication $\tilde{R}_{l+1} Q(l+2, l+1)$ can be handled in two different ways [18, 19]. First, $Q(l+2, l+1)$ may be generated from the Kronecker representation as a sparse matrix and the premultiplication with $\tilde{R}_{l+1}$ performed. Second, the efficient vector–Kronecker product algorithm [101, 136] can be used to multiply rows of $\tilde{R}_{l+1}$ with the

ALGORITHM 9. *Matrix analytic computation of steady-state vector of an LDQBD.*

Choose a suitable Lyapunov function $g(\mathbf{i})$ proving ergodicity:
  Choose $g(\mathbf{i})$ such that the set of states for which $g(\mathbf{i}) < \infty$ is finite;
  Obtain the drift $d(\mathbf{i})$ and show that it is bounded;
  $\chi := \sup_{\mathbf{i}\in\mathcal{R}} d(\mathbf{i})$ (using HOM4PS2-2.0 if necessary);
  $\gamma := \chi(1/\varepsilon - 1)$ for given ε;
  Show that $\mathcal{C} = \{\mathbf{i} \in \mathcal{R} \mid d(\mathbf{i}) > -\gamma\}$ is finite;
Compute $(Low, High)$;
$\tilde{R}_{High} := 0$;
For $l := High - 1, \ldots, Low$,
  $A_l := Q(l+1, l+1) + \tilde{R}_{l+1}Q(l+2, l+1)$;
  $B_l := -Q(l, l+1)$;
  Solve $\tilde{R}_l A_l = B_l$ for $\tilde{R}_l$;
$A_{Low-1} := Q(Low, Low) + \tilde{R}_{Low}Q(Low+1, Low)$;
Solve $\tilde{\boldsymbol{\pi}}(\mathcal{R}_{Low})A_{Low-1} = \mathbf{0}$ for $\tilde{\boldsymbol{\pi}}(\mathcal{R}_{Low})$ subject to $\|\tilde{\boldsymbol{\pi}}(\mathcal{R}_{Low})\|_1 = 1$;
For $l := Low, \ldots, High - 1$,
  $\tilde{\boldsymbol{\pi}}(\mathcal{R}_{l+1}) := \tilde{\boldsymbol{\pi}}(\mathcal{R}_l)\tilde{R}_l$;
  Normalize $\tilde{\boldsymbol{\pi}}$ subject to $\|(\tilde{\boldsymbol{\pi}}(\mathcal{R}_{Low}), \ldots, \tilde{\boldsymbol{\pi}}(\mathcal{R}_{l+1})\|_1 = 1$;
Compute $\|\boldsymbol{\pi}Q\|_1$ by letting
  $\boldsymbol{\pi} := (\mathbf{0}(\mathcal{R}_0)^T, \ldots, \mathbf{0}(\mathcal{R}_{Low-1})^T, \tilde{\boldsymbol{\pi}}(\tilde{\mathcal{R}}), \mathbf{0}(\mathcal{R}_{High+1})^T, \ldots)$.

block $Q(l+2, l+1)$, while the latter operand is being held in Kronecker form. 1-norm of the residual vector of the countably infinite model [112] is computed from

$$\|\boldsymbol{\pi}Q\|_1 := \sum_{\mathbf{i}\in\tilde{\mathcal{R}}} \left|\tilde{\mathbf{r}}(\mathbf{i}) - \sum_{\mathbf{j}\notin\tilde{\mathcal{R}}} \tilde{\boldsymbol{\pi}}(\mathbf{i})Q(\mathbf{i},\mathbf{j})\right| + \sum_{\mathbf{i}\in\tilde{\mathcal{R}}}\sum_{\mathbf{j}\notin\tilde{\mathcal{R}}} \tilde{\boldsymbol{\pi}}(\mathbf{i})Q(\mathbf{i},\mathbf{j}) \quad \text{with} \quad \tilde{\mathbf{r}} := \tilde{\boldsymbol{\pi}}\tilde{Q},$$

where $\tilde{Q}$ is the truncated generator matrix. Once $(Low, High)$ is determined, computation of the steady-state vector of the LDQBD can also be performed using (block) GE [125, 158, 302, 305]. This approach can be equally efficient [166], but unfortunately does not provide the $\tilde{R}_l$ matrices.

Having observed in [107, 120] that the $\tilde{R}_l$ matrices are not necessarily sparse, their full and sparse storages are considered. When the $\tilde{R}_l$ matrices are stored as full matrices, a temporary matrix in full storage needs to be set aside to form A_l and then compute its LU factorization in place. Since $\tilde{R}_{High} := 0$, the sparsity pattern of A_{High-1} is equal to that of $Q(High, High)$. Therefore, $\tilde{R}_{High-1}$ should be obtained using sparse LU factorization even though it is later stored as a full matrix. Now, although all $\tilde{R}_l$ matrices are to be kept until the computation ends to obtain the steady-state solution and the sizes of A_l and $\tilde{R}_l$ matrices are known a priori at each level, it is still possible to consider two different memory allocation–deallocation schemes for these two matrices [18, 19]. First, memory to store $\tilde{R}_l$ matrices from $l = High - 1$ down to Low

and the largest temporary matrix, A_{High-1}, can be allocated at the outset and deallocated at the end of the computation. In this scheme, successive A_l matrices overwrite the same temporary matrix. Second, the memory of A_l can be deallocated at the end of step l, and the memory for $\tilde{R}_{l-1}$ and A_{l-1} can be allocated at the beginning of step $(l-1)$. When the temporary matrix is allocated once at the outset, peak total memory usage turns out to be higher especially when H_I, number of countably infinite dimensions, is larger. On the other hand, when $\tilde{R}_l$ matrices are chosen to be stored as sparse matrices, the temporary matrix to form A_l is also stored as a sparse matrix, and memory is allocated at each level. This requires the allocation of extra memory for the sparse LU factorization of A_l due to expected fill-in.

In the proposed Kronecker representations in Chapter 4 for countably infinite block tridiagonal generator matrices, values of nonzero entries of submodel transition matrices should be available during computation. In many cases, submodel transition matrices have subdiagonal, diagonal, or superdiagonal nonzero structures. If two submodel transition matrices associated with the same transition class have the same nonzero structure, then it is possible for the two matrices to share the storage space of one vector. In this case, a vector of size equal to the larger state space size of the two submodels is to be allocated. Besides, when $X_{0,1}^{(h)} = 1$ and $X_{p,l}^{(h)} = 1$ for $h = 1, \ldots, H$, $l = 1, \ldots, H_I$, and $p > Low$ as in some models, memory required to store nonzero entries of submodel transition submatrices at a given level is smaller than that of a higher level. In this case, it is feasible to allocate memory to store submatrices once at the highest level and keep reusing it when moving from level $High$ down to Low. Otherwise, memory necessary to store nonzero entries may be allocated and deallocated on the fly. Furthermore, vector–Kronecker product multiplication requires an additional vector over vector–matrix multiplication. When $\tilde{R}_l$ and A_l are stored as sparse matrices, an additional temporary vector is used to compute, compact, and store the rows of A_l. Besides, adding a row of a matrix in Kronecker form to a vector requires two additional vectors. We also choose to store the values of transition rate functions for states in levels $l-1$,l,$l+1$ when processing level l in order not to evaluate the functions multiple times. We allocate all the additional vectors at the outset and deallocate them at the end. Amount of memory allocated for all these vectors is negligible compared to the total amount of memory allocated for $\tilde{R}_l$ matrices.

If memory is at a premium, one can also consider the more recent approach in [21] that reduces memory requirements further by enabling the computation of steady-state expectations without having to obtain the steady-state distribution. The approach is inspired by a Horner-like computational scheme in which only the conditional expected sojourn time matrix $\tilde{R}_l$ at level l needs to be allocated in step l; otherwise, there are no time savings obtained. In other words, not all $\tilde{R}_l$ matrices for $l = High - 1$ down to Low need to be stored simultaneously. In order to evaluate M different functions of the steady-state distribution, $(M+1)$ temporary vectors of size equal to the num-

ber of states within levels *Low* through *High* must be used. For instance, if the first moment (i.e., mean) is to be computed for $M = H_I$ countably infinite variables, $(H_I + 1)$ temporary vectors of length $|\tilde{\mathcal{R}}|$ must be allocated. The additional vector is used for normalization purposes. At step l, $\tilde{R}_l$ is computed as usual and stored. This implies that $\tilde{R}_{l+1}$ from the previous step need not be in memory any longer, and hence, $\tilde{R}_l$ is the only conditional expected sojourn time matrix in memory at step l in this approach. Then $\tilde{R}_l$ is multiplied with the subvectors corresponding to level $(l+1)$ of the $(M+1)$ temporary vectors. The product is added to the running sum of subvectors corresponding to level l of the $(M + 1)$ temporary vectors in order to keep on accumulating steady-state expectations. With this memory efficient approach, steady-state measures based on average costs or rewards, moments, and cumulants can be computed. As shown in [18, 19], memory savings can be substantial with the approach in [21] in some models. This alternative solver is able to compute mean values of variables stably as the solver in Algorithm 9. The reported relative errors of the mean values obtained with the memory efficient solver with respect to those obtained with the original solver are close to machine precision in all cases. However, there is one drawback; it is the absence of the accuracy measure, $\|\boldsymbol{\pi}Q\|_1$, because the steady-state distribution, $\boldsymbol{\pi}$, is no longer available.

In between full and sparse storages of $\tilde{R}_l$ matrices, full storage is better in models having very high nonzero densities in the $\tilde{R}_l$ matrices. When there are memory savings with sparse storage of $\tilde{R}_l$ matrices [18, 19], the respective time savings are even more substantial. The temporary matrix A_l seems to be benefiting considerably from sparse LU factorization. On the other hand, there is observable difference between using sparse versus Kronecker representations of the $Q(l+1, l)$ matrices. This is the case because each subdiagonal nonzero block is used once, and the sparse generation procedure associated with it and the pre-multiplication with $\tilde{R}_{l+1}$ amount to performing the same number of flops as would be done by the vector–Kronecker product multiplication algorithm between the rows of $\tilde{R}_{l+1}$ and the subdiagonal nonzero block when the latter is kept in Kronecker form.

The scalability of different LDQBD solver implementations for increasing values of ε has been investigated in [18]. Since time complexity of A_l's LU factorization at level l is cubic in the order of $|\mathcal{R}_l|$ for dense $\tilde{R}_l$ matrices and $|\mathcal{R}_l|$ is a polynomial with degree $(l-1)$, time requirements become more pronounced for models with higher H_I values. The situation regarding memory is better since the requirements at level l are quadratic in $|\mathcal{R}_l|$ for dense $\tilde{R}_l$ matrices. Clearly, time and memory requirements are much better when the $\tilde{R}_l$ matrices are sparser. Given more memory and time, it is always possible to obtain more accurate results with the matrix analytic approach. Dropping nonzeros less than 10^{-16} from matrices is observed not to have an adverse effect on accuracy but likely to help with memory requirements [120].

In [103], an alternative technique for systems of stochastic chemical kinetics which is also based on using Lyapunov functions is investigated. The

technique explores states that are only in the set concentrating the steady-state probability mass and then resorts to polyhedra theory [86] to bound steady-state probabilities. Although the technique can potentially work on states arbitrarily far away from the origin, it yields results that are much less accurate compared to those provided by the technique discussed here.

In closing this section, we remark that the MAP/PH/S queueing system with acyclic PH retrials [112] considered in Chapter 4 could also be modeled as an LDQBD by choosing an appropriate level definition. However, the truncated model is represented as a block matrix with blocks of equal size using sums of Kronecker products, and its steady-state solution is computed iteratively with SOR. The reason for this choice is that the matrix analytic method requires the computation of the matrices of conditional expected sojourn times, and this computation does not scale well as the number of dimensions in the multidimensional MC increases. This is due to the increase in the order of the diagonal blocks as the LDQBD level number increases in multidimensional MCs. An implementation within the `NSolve` package of the APNN Toolbox [7, 22] is available at [114].

6.6 Working with Compact Solution Vectors

The HTD format for compact vectors discussed in the previous chapter is recently employed in power, JOR, and GMRES iterative methods using two adaptive truncation strategies for the solution vectors of MCs [66]. The performance of the resulting iterative solvers are compared on a large number of multidimensional problems, two of which are the availability and polling models in Chapter 2, with their Kronecker structured counterparts that employ full solution vectors of length $|\mathcal{R}|$, the size of the reachable state space. In this section, we briefly review the outcome of the study in [66].

Let us recall that the power method is successfully employed in the *PageRank* algorithm [41] for Google matrices [106], and JOR is essentially a preconditioned power method in which the preconditioner is $M := Q_D/\omega$ and, hence, diagonal. The convergence rate of these methods is known to depend on the magnitude of the subdominant eigenvalue of the corresponding iteration matrix. In the original PageRank algorithm, this value is set to 0.85 by construction to guarantee a certain convergence rate. Furthermore, JOR is known to converge for $\omega \in (0, 1)$, that is, under-relaxation, as discussed in Section 6.1. On the other hand, GMRES is a projection method which extracts solutions from an increasingly higher-dimensional Krylov subspace by enforcing a minimality condition on the residual 2-norm at the expense of having to compute and store a new *orthogonal basis vector* for the subspace at each iteration. This orthogonalization is accomplished through what is called an *Arnoldi process*. In theory, GMRES converges to the solution in at most $|\mathcal{R}|$ iterations. However, this may become prohibitively expensive at least in

terms of space, so in practice, a restarted version with a finite subspace size of m is to be used. Hence, the number of vectors allocated for the representation of the Krylov subspace in the *restarted* GMRES solver will be limited by m. If the vectors are stored in compact format, m can be set to a larger value without exceeding the available memory which is expected to improve the convergence of GMRES.

Unlike the power and GMRES methods, the iteration vector also needs to be multiplied by the reciprocated diagonal elements of the coefficient matrix in the JOR method. Since it is costly to generate each element of a vector in HTD format, an iterative method such as *Newton–Schulz* [202] can be used to perform the elementwise reciprocation of the diagonal vector in HTD format at the outset using Algorithm 10.

ALGORITHM 10. *Elementwise reciprocation of vector* $\mathbf{d}$ *as* $\mathbf{dinv}$.

$\mathbf{dinv} := \mathbf{e}^T/\|\mathbf{d}\|_2$; $it := 0$; $nrm := 0$; $nrm\_chg := 0$; $rcp\_stop := FALSE$;
While $nrm \geqslant$ `rcp_tol` and $rcp\_stop = FALSE$,
    $it := it + 1$;
    $\mathbf{dinv}' := \mathbf{dinv}$;
    $\mathbf{\Delta d} := \mathbf{e}^T - \mathbf{d} \star \mathbf{dinv}'$;
    $nrm' := \|\mathbf{\Delta d}\|_2/\|\mathbf{e}\|_2$;
    $\mathbf{dinv} := \mathbf{dinv}' + \mathbf{dinv}' \star \mathbf{\Delta d}$;
    If $it > 1$,
        $nrm\_chg := nrm'/nrm$;
    If $nrm\_chg >$ `rcp_chg_tol` and $it \geqslant$ `rcp_maxit`,
        $rcp\_stop := TRUE$;
    Else
        $nrm := nrm'$.

The algorithm starts by initializing $\mathbf{dinv}$ (which becomes $\mathbf{dinv}'$ in the current iteration) with a vector whose entries are equal to each other in the same vein as the initialization of the starting vector in iterative methods with the uniform distribution. This step requires the computation of the 2-norm of $\mathbf{d}$. The other operations that the algorithm executes are elementwise multiplication of two vectors which is denoted by the operator $\star$, addition of two vectors, and computation of the 2-norm of vectors. The algorithm stops if the 2-norm of $(\mathbf{e}^T - \mathbf{d} \star \mathbf{dinv}')$ relative to the 2-norm of $\mathbf{e}$ is smaller than `rcp_tol` [202] or if this relative norm divided by the relative norm in the previous iteration is greater than or equal to `rcp_chg_tol` when `rcp_maxit` iterations have already been performed. We remark that the Newton–Schulz method is a nonlinear iteration with quadratic convergence rate [185].

Among the operations needed to implement Algorithm 10, elementwise multiplication of two compact vectors in HTD format has not been discussed in Section 5.3. Note that this operation is required not only in the Newton–Schulz iteration for computing the reciprocated diagonal elements in $\mathbf{dinv}$ at

the outset but also in multiplying **dinv** with the updated solution vector at each JOR iteration. Let us now explain how this operation can be performed.

Elementwise multiplication of two matrices Y and X with SVDs $Y = U_Y \Sigma_Y V_Y^T$ and $X = U_X \Sigma_X V_X^T$ yields

$$Y \star X := (U_Y \odot^T U_X)(\Sigma_Y \otimes \Sigma_X)(V_Y \odot^T V_X)^T,$$

where $\odot^T$ denotes a transposed variant of the *Khatri–Rao product* [199]. More specifically, the ith row of the $(n_{AB} \times r_A r_B)$ matrix $A \odot^T B$ is the Kronecker product of the ith rows of the $(n_{AB} \times r_A)$ matrix A and the $(n_{AB} \times r_B)$ matrix B. This implies that elementwise multiplication $Y \star X$ has rank equal to the product of the ranks of Y and X. Elementwise multiplication of two vectors $\mathbf{y}$ and $\mathbf{x}$ in HTD format is no different if SVD is replaced with HOSVD. The operation can be carried out in four steps [202], which are orthogonalization of the vectors in HTD format, computation of their Gramians, computation of SVDs of Gramians, and update of basis and transfer matrices. There is also a truncation step that needs to be performed when the elementwise multiplication is extracted from the implicitly formed Kronecker product by again imposing an accuracy of `trunc_tol` on the truncated HOSVD.

The elementwise reciprocation of $\mathbf{d}$ using Newton–Schulz introduces two different truncations [202], first, before Newton–Schulz, when the HTD of the diagonal elements in $\mathbf{d}$ is computed, and, second, during the iterative computation of the reciprocated diagonal elements in **dinv** within Newton–Schulz. Since a numerical iterative method is employed to compute the reciprocal values, it is not clear how fast convergence takes place in practice since the initial transient period that is needed for the asymptotic quadratic convergence behavior to manifest itself can be time consuming [185]. Therefore, an alternative approach is also considered in [66].

The alternative approach is to take advantage of identical entries in $\mathbf{d}(\mathcal{R}_p)$ for $p = 0, \ldots, N-1$. In many multidimensional Markovian models, there are entries with the same value in $\mathbf{d}(\mathcal{R}_p)$ because the CTMC results from some compact model specification which contains only a few parameters compared to the size of the reachable state space, $\mathcal{R}$. This may be exploited by defining an equivalence relation among the states in $\mathcal{R}_p^{(h)}$ for $h = 1, \ldots, H$ by using row sums of the submatrices $Q_k^{(h)}(\mathcal{R}_p, \mathcal{R}_w)$ that contribute to sums of Kronecker products for $k \in \mathcal{K}_{p,w}$ and $w = 0, \ldots, N-1$. Then the states in $\mathcal{R}_p$ that are in the same equivalence class in all dimensions have identical diagonal entries in $\mathbf{d}(\mathcal{R}_p)$ [66]. This approach can be very effective in representing $\mathbf{d}(\mathcal{R}_p)$ compactly when the number of equivalence classes in each dimension is small. All that needs to be done is to represent the reciprocated diagonal entries in each equivalence class using a Kronecker product, which in turn can be represented in HTD format. Thus, for each combination of equivalence classes across all dimensions, an HTD formatted vector must be constructed. Then all such vectors need to be added and truncated so that the reciprocated diagonal elements in $\mathbf{dinv}(\mathcal{R}_p)$ are represented by a single HTD formatted

vector. In the worst case, one vector in HTD format needs to be constructed for each state meaning all entries in $\mathbf{d}(\mathcal{R}_p)$ are different from each other. However, for many models, the number of HTD formatted vectors to be added and truncated is much smaller.

In `htucker`, memory requirements of the HTD format are limited by a fixed truncation error tolerance, `trunc_tol`, and a fixed rank bound, `max_rank`, for the truncated HOSVD. In many cases, this approach results in ranks becoming large during the first few iterations when the iteration vector is not close to the solution. Furthermore, imposing a fixed rank bound may also limit the accuracy of the final solution that can be obtained. Therefore, it is more efficient to use a larger `trunc_tol` when the 2-norm of the residual vector, $\|\mathbf{r}_{(it)}\|_2$, is large and to decrease `trunc_tol` when $\|\mathbf{r}_{(it)}\|_2$ is becoming small. Thus, an *adaptive strategy* for adjusting the truncation error is required, and two such strategies are considered in [66].

In the first adaptive strategy named AS1, `trunc_tol`, is initially set to $\texttt{trunc\_tol} := \texttt{stop\_tol}/\sqrt{2H-3}$ and then updated when $\|\mathbf{r}_{(it)}\|_2$ is computed at iteration *it* as

$$\texttt{prev\_trunc\_tol} := \texttt{trunc\_tol},$$
$$\texttt{trunc\_tol} := \max(\min(\texttt{prev\_trunc\_tol}, \sqrt{\|\mathbf{r}_{(it)}\|_2}\ \texttt{stop\_tol}/\sqrt{2H-3}), 10^{-16}).$$

Initially, $\texttt{prev\_trunc\_tol} := 0$. In this way, `trunc_tol` is made to remain within $(10^{-16}, \texttt{prev\_trunc\_tol})$ while being forced to decrease conservatively with a decrease in $\|\mathbf{r}_{(it)}\|_2$.

In the second adaptive truncation strategy named AS2, we start with the initialization $\texttt{prev\_trunc\_tol} := 0$ and $\texttt{trunc\_tol} := 100\ \texttt{stop\_tol}/\sqrt{2H-3}$. Then we update the two variables as

$$\texttt{prev\_trunc\_tol} := \texttt{trunc\_tol},$$
$$\texttt{trunc\_tol} := \max(\min(\texttt{prev\_trunc\_tol}, \sqrt{\|\mathbf{r}_{(it)}\|_2}\ 10^{-4}/\sqrt{2H-3}), 10^{-16}).$$

when $\|\mathbf{r}_{(it)}\|_2$ is computed at iteration *it*. AS2 starts at a larger `trunc_tol` and is expected to increase `trunc_tol` more conservatively compared to AS1 and independently of the value of `stop_tol`.

The scalability of the HTD format in the compact vector iterative solvers of power, JOR, and GMRES using the above adaptive truncation strategies is investigated on three models for increasing sizes of the reachable state space. Two of the models are the availability and polling models in Chapter 2, and the third one is a cloud computing model from [154]. First, we summarize the effect of using the Newton–Schulz iteration and the equivalence class approach to compute **dinv** at the outset for JOR.

Each of the availability and cloud computing models has a single reachable state space partition and ends up enumerating a large percentage of the entries in **d** to construct **dinv**, since the number of equivalence classes in dimension h of both models is relatively large with respect to the state space size $|\mathcal{S}^{(h)}|$ for $h = 1, \ldots, H$. It is observed that the time to compute **d** and therefore **dinv** in HTD format grows quickly for larger values of H in these two models using both approaches. However, the Newton–Schulz approach can be made faster by limiting `rcp_maxit` in Algorithm 10. At the moment, it starts with truncated HOSVD ranks of 1 and increases the ranks when needed as the iterations progress. For these two models, the equivalence class approach for computing **dinv** in HTD format yields a more compact representation than the Newton–Schulz approach. As for the polling model which has multiple reachable state space partitions, the number of equivalence classes used to represent **d** is small. Hence, this approach yields not only the more compact representation but also the faster one. In this case, the HTD representation of **dinv** remains compact even for larger values of H, with its memory requirements growing more or less linearly in H.

Solution vectors of the Kronecker-based solvers power and JOR using the HTD format cease to be unit 1-norm due to truncation with `trunc_tol` and are normalized at each iteration. The check on `stop_tol` is also performed at each iteration in this case. For Kronecker-based power and JOR solvers with full vectors, normalization and the stopping test can take place every some number of (e.g., 10) iterations. The parameters of solvers used in the experiments are $\Delta := 0.999/\max_{i \in \mathcal{R}} |q_{i,i}|$ for power, $\omega := 0.75$ for JOR, $m := 30$ for the Krylov subspace size of GMRES, `stop_tol` $:= 10^{-8}$, and `max_time` := 1,000 seconds. Detailed results can be found in [66].

Memory requirements of Kronecker-based full-vector solvers can be calculated at the outset. Power and JOR each require three vectors of length $|\mathcal{R}|$ (for Q_D, $\boldsymbol{\pi}_{(it)}$, and $\boldsymbol{\pi}_{(it+1)}$) and two vectors of length $\max_p |\mathcal{R}_p|$ (for the shuffle algorithm to carry out the full vector–Kronecker product multiplication), whereas restarted GMRES(m) requires $(m + 3)$ vectors of length $|\mathcal{R}|$ and two vectors of length $\max_p |\mathcal{R}_p|$. Clearly, largest versions of the three models cannot be handled with full-vector solvers on a platform having 16 GB of main memory. Similarly, the Kronecker-based full-vector GMRES solver cannot be utilized in the six-dimensional polling model. For Kronecker-based compact vector solvers, it is not possible to forecast memory requirements at the outset when adaptive truncation strategies are used. This is because the maximum number of floating-point array elements allocated to matrices in the HTD format representing vectors and the workspace used in the solution process depends on the values in the vectors that are represented compactly at each iteration, and, hence, the character of the particular model and the behavior of the solver, and also on the value of `trunc_tol`.

There are cases where we have not been able to obtain **dinv** in HTD format for JOR(0.75) within `max_time` or due to memory limitations sometimes with both computational approaches and sometimes with Newton–Schulz alone.

On the other hand, there is a case where the Kronecker-based compact vector power method with AS2 exits due to a non-converging `LAPACK` method that is used to compute SVD. Similarly, there are a few cases where we have not been able to obtain results with the Kronecker-based compact vector GMRES(30) solver using AS1. The solver performs a number of Arnoldi process steps and then takes an exceedingly large amount of time without producing a result and therefore is aborted. Finally, there is a case where the Kronecker-based compact vector JOR(0.75) solver with AS1 using the equivalence class approach for reciprocation fails due to memory limitations.

Among the Kronecker-based full-vector solvers for all three models, JOR is almost always the fastest solver yielding the smallest $\|\mathbf{r}_{(it)}\|_2$ with the smallest memory, which is about one order of magnitude smaller than what is required by GMRES. Among the compact vector solvers, those that use AS1 never require a larger number of iterations than their counterparts with AS2 for the same $\|\mathbf{r}_{(it)}\|_2$. The average time per iteration of compact vector solvers with AS2 is lower than that with AS1. However, power method with AS2 stagnates and is not able to meet `stop_tol` in any of the problems. The convergence behavior of compact vector solvers employing AS2 tend to be more unpredictable and less satisfactory than that with AS1. Whenever a compact vector solver using AS1 stops, $\|\mathbf{r}_{(it)}\|_2 < \texttt{stop\_tol}$, whereas this inequality in general does not hold for AS2. Regarding memory requirements however, AS2 is better, sometimes by several orders of magnitude, than the respective solver that uses AS1. One would expect this to imply that as the number of dimensions increases, a larger number of iterations can be performed by AS2 in the same duration, thereby bringing the solver closer, if not, to convergence. This seems to be the case especially when the decrease in memory consumption is substantial. The full-vector approach is faster for smaller models, but it is outperformed by the compact vector approach for larger models especially when it is used with the power method and AS2.

The availability model has highly unbalanced transition rates due to infrequent failures. Failures correspond to local transitions, and since they are infrequent, the system is available most of the time, and the steady-state distribution is skewed. Effects of varying the original transition rates given in Section 5.3 for $H = 6$ are investigated on two variants. The first variant has one tenth, and the second variant has ten times the failure rates of the original model. The steady-state distribution becomes more skewed in the first variant since the system is even more available in the long run. This translates into a less difficult problem to solve, meaning it takes a smaller number of iterations for the same accuracy of the solution. The results show that indeed all solvers are sensitive to the transition rates in the model. Memory requirements and computation time of **dinv** increase in the more difficult variant. Memory requirements of power and JOR(0.75) compact solvers increase when the problem becomes more difficult, resulting in longer time per iteration and a smaller number of iterations in the same time duration.

In summary, the compact vector approach when coupled with an adaptive truncation strategy is memory and relatively time efficient, having the potential to increase the size of solvable models on a given platform significantly, and therefore deserves further investigation.