

ON THE NUMERICAL ANALYSIS OF INFINITE MULTI-DIMENSIONAL MARKOV CHAINS

A DISSERTATION SUBMITTED TO
THE GRADUATE SCHOOL OF ENGINEERING AND SCIENCE
OF BILKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR
THE DEGREE OF
DOCTOR OF PHILOSOPHY
IN
COMPUTER ENGINEERING

By
Muhsin Can Orhan
July 2017

ON THE NUMERICAL ANALYSIS OF INFINITE MULTI-
DIMENSIONAL MARKOV CHAINS

By Muhsin Can Orhan

July 2017

We certify that we have read this dissertation and that in our opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Tuğrul Dayar(Advisor)

Nail Akar

Murat Manguoğlu

Selim Aksoy

Mehmet Halit Seyfullah Oğuztüzün

Approved for the Graduate School of Engineering and Science:

Ezhan Karaşan
Director of the Graduate School

ABSTRACT

ON THE NUMERICAL ANALYSIS OF INFINITE MULTI-DIMENSIONAL MARKOV CHAINS

Muhsin Can Orhan
Ph.D. in Computer Engineering
Advisor: Tuğrul Dayar
July 2017

A system with multiple interacting subsystems that exhibits the Markov property can be represented as a multi-dimensional Markov chain (MC). Usually the reachable state space of this MC is a proper subset of its product state space, that is, Cartesian product of its subsystem state spaces. Compact storage of the infinitesimal generator matrix underlying such a MC and efficient implementation of analysis methods using Kronecker operations require the set of reachable states to be represented as a union of Cartesian products of subsets of subsystem state spaces.

We first show that the problem of partitioning the reachable state space of a three or higher dimensional system with a minimum number of partitions into Cartesian products of subsets of subsystem state spaces is NP-complete. Two algorithms, one merge based the other refinement based, that yield possibly non-optimal partitionings are presented. Results of experiments on a set of problems from the literature and those that are randomly generated indicate that, although it may be more time and memory consuming, the refinement based algorithm almost always computes partitionings with a smaller number of partitions than the merge based algorithm.

When the infinitesimal generator matrix underlying the MC is represented compactly using Kronecker products, analysis methods based on vector-Kronecker product multiplication need to be employed. When the factors in the Kronecker product terms are relatively dense, vector-Kronecker product multiplication can be performed efficiently by the shuffle algorithm. When the factors are relatively sparse, it may be more efficient to obtain nonzero elements of the generator matrix in Kronecker form on-the-fly and multiply them with corresponding elements of the vector. We propose a modification to the shuffle algorithm that multiplies relevant elements of the vector with submatrices of factors in which zero rows and columns are omitted. This approach avoids unnecessary floating-point

operations that evaluate to zero during the course of the multiplication. Numerical experiments on a large number of models indicate that, in many cases the modified shuffle algorithm performs a smaller number of floating-point operations than the shuffle algorithm and the algorithm that generates nonzeros on-the-fly, sometimes with minimum number of floating-point operations and amount of memory possible.

Although the generator matrix is stored compactly using Kronecker products, solution vectors used in the analysis still require memory proportional to the size of the reachable state space. This becomes a bigger problem as the number of dimensions increases. We show that it is possible to use the hierarchical Tucker decomposition (HTD) to store the solution vectors during Kronecker-based Markovian analysis relatively compactly and still carry out the basic operation of vector-matrix multiplication in Kronecker form relatively efficiently.

The time evolution of a stochastic chemical system modelled as a continuous-time MC (CTMC) can be described as a system of ordinary differential equations (ODEs) known as the chemical master equation (CME). The CME can be analyzed by discretizing time and solving a linear system obtained by truncating the countably infinite state space at each time step. However, it is not trivial to choose a truncated state space that includes few states with negligible probabilities and leaves out only a small probability mass. We show that it is possible to decrease the memory requirement of the ODE solver using HTD with adaptive truncation strategies and we propose a novel approach to truncate the countably infinite state space using prediction vectors. Numerical experiments indicate that adaptive truncation strategies improve time and memory efficiency significantly when fixed truncation strategies are inefficient.

Finally, we consider a multi-class multi-server retrial queueing system in which customers arrive according to a class-dependent Markovian arrival process (MAP). Service and retrial times follow class-dependent phase-type (PH) distributions with the further assumption that PH distributions of retrial times are acyclic. Here, we obtain a necessary and sufficient condition for ergodicity from criteria based on drifts. The countably infinite state space of the model is truncated with an appropriately chosen Lyapunov function. The truncated model is described as a multi-dimensional MC and a Kronecker representation of its infinitesimal generator matrix is numerically analyzed.

Keywords: Markov chain, Kronecker representation, Cartesian product partitioning, vector-Kronecker product multiplication, hierarchical Tucker decomposition,

chemical master equation, retrial queues.



ÖZET

SONSUZ ÇOK BOYUTLU MARKOV ZİNCİRLERİNİN SAYISAL ÇÖZÜMLEMESİ ÜZERİNE

Muhsin Can Orhan

Bilgisayar Mühendisliği, Doktora

Tez Danışmanı: Tuğrul Dayar

Temmuz 2017

Birden fazla alt sistemden oluşan ve Markov özelliğine sahip bir sistemi çok boyutlu bir Markov zinciri olarak göstermek mümkündür. Bu Markov zincirinin ulaşılabilir durum uzayı, genellikle çarpım durum uzayının bir öz alt kümesidir. Böyle bir Markov zincirine karşı gelen matrisin sıkıştırılmış olarak saklanması ve Kronecker işlemlerin etkili bir şekilde gerçekleştirilebilmesi, ulaşılabilir durum uzayının alt sistem durum uzaylarının alt kümelerinin Kartezyen çarpımlarının birleşimi olarak yazılmasını gerektirir.

Bu çalışmada ilk olarak, üç veya daha fazla boyutlu bir sistemin ulaşılabilir durum uzayının en az sayıda alt sistem durum uzaylarının alt kümelerinin Kartezyen çarpımlarına bölünme probleminin, NP-tam bir problem olduğunu gösteriyoruz. Bu problemi çözmek için, eniyi çözümü bulması kesin olmayan, biri birleştirme ve diğeri geliştirme temelli olmak üzere iki algoritma öneriyoruz. Literatürde yer alan ve rasgele olarak oluşturulmuş örneklerle yaptığımız deneyler, geliştirme temelli algoritmanın daha çok zaman ve bellek gerektirdiğini, ancak neredeyse her zaman daha az sayıda bölünme bulabildiğini gösteriyor.

Markov zincirine karşılık gelen matris, Kronecker çarpımları kullanılarak sıkıştırılmış olarak gösterildiğinde, özümleme yöntemleri vektör-Kronecker terim çarpımının üzerine bina edilir. Kronecker terimlerdeki çarpan matrisler yoğun olduğunda, karma algoritması vektör-Kronecker terim çarpımını etkili bir şekilde hesaplayabilir. Çarpan matrisler seyrek olduğunda ise, Markov zincirine karşılık gelen matrisin sıfırdan farklı elemanlarını çarpım esnasında oluşturup vektörde karşılık gelen elemanlarla çarpmak daha etkili olabilmektedir. Karma algoritmasını, Kronecker terimin çarpan matrislerindeki tamamı sıfır olan satır ve sütunları gözardı edecek şekilde değiştirmeyi öneriyoruz. Bu değişiklik, sonucu sıfır olan kayan noktalı sayı işlemlerinin gerçekleştirilmemesini sağlıyor. Çok sayıda modelde, değiştirilmiş karma algoritması pek çok kayan noktalı sayı

işleminden kaçınılmasını sağlıyor. Yine bazı modellerde, değiştirilmiş karma algoritmasının sıfırdan farklı elemanları çarpım esnasında oluşturan algoritmaya kıyasla daha az kayan noktalı sayı işlemi ve bellek gerektiriyor.

Markov zincirine karşılık gelen matris, Kronecker çarpımları kullanılarak sıkıştırılmış olarak gösterilse bile, işlemler için gereken bellek miktarı ulaşılabilir durum uzayının büyüklüğüyle doğru orantılı olarak değişmektedir. Özellikle boyut sayısı arttığında, bu daha büyük bir sorun oluşturmaktadır. Hiyerarşik Tucker ayrıştırması kullanarak, çözüm vektörlerinin görece sıkıştırılmış olarak tutulup, temel vektör–Kronecker terim çarpma işlemlerinin görece etkili olarak gerçekleştirilebildiğini gösteriyoruz.

Sürekli zamanlı bir Markov zinciri olarak modellenmiş bir rassal kimyasal sistemin zamana bağlı değişimi kimyasal ana denklemi olarak da bilinen bir adi diferansiyel denklem sistemi olarak tanımlanabilir. Kimyasal ana denklemi, zamanı ayrıklaştırarak ve sayılabilir sonsuz durum uzayının kesilerek elde edilmiş doğrusal sistemin çözülmesiyle çözümlenebilir. Durum uzayının ihmal edilebilir az sayıda durum içererek, kesilmiş durum uzayı dışında az miktarda olasılık kitlesi kalacak şekilde kesilmesi ise apaçık değildir. Hiyerarşik Tucker ayrıştırması kullanarak adi diferansiyel denklem çözücüsünün ihtiyaç duyduğu bellek miktarını azaltılabileceğini gösteriyoruz. Ayrıca, sonsuz durum uzayını kesmek için tahmin vektörlerini kullanan yenilikçi bir yöntem öneriyoruz. Sayısal deneyler uyarlamalı kesim stratejilerinin zaman ve bellek ihtiyacını, sabit kesim stratejilerine kıyasla ciddi olarak azalttığını gösteriyor.

Son olarak, birden fazla sınıfa ait müşteri kabul eden, müşterilerin sınıfa bağımlı Markov varış sürecine göre geldikleri ve çok sayıda sunucusu olan bir yeniden denemeli kuyruk sistemini ele alıyoruz. Ele aldığımız sistemde, servis zamanlarının sınıfa bağımlı evre-tipli, yeniden deneme zamanlarının ise çevrimsiz sınıfa bağımlı evre-tipli dağılım gösterdiklerini farz ediyoruz. Bu sistemin ölçüm kalışının gerek ve yeter koşulunu ise sürüklenme fonksiyonlarına dayanan ölçütler kullanarak elde ediyoruz. Sonsuz durum uzayını uygun bir şekilde seçilmiş Lyapunov fonksiyonu kullanarak kesiyoruz. Elde ettiğimiz kesilmiş modeli ise çok boyutlu bir Markov zincir olarak tanımlayıp bu zincire karşılık gelen matrisin Kronecker temelli sayısal çözümlemesini gerçekleştiriyoruz.

Anahtar sözcükler: Markov zinciri, Kronecker gösterimi, Kartezyen çarpım bölümlenmesi, vektör–Kronecker terim çarpımı, hiyerarşik Tucker ayrıştırması, kimyasal ana denklemi, yeniden denemeli kuyruklar.

Acknowledgement

I am grateful to my advisor Tuğrul Dayar for his wisdom, patience, and guidance. I am inspired by his dedication to science and learned a lot about being a good researcher from him. I thank the members of my Ph.D. advisory committee members Nail Akar and Murat Manguoğlu for their valuable suggestions, and my Ph.D. jury members Selim Aksoy and Halit Oğuztüzün for their comments on this thesis. I am grateful to my family for their encouragement and support. Finally, I also thank TÜBİTAK for its financial support during my Ph.D. study and the Alexander von Humboldt Foundation Research Group Linkage Programme for its support of the work in Chapter 4 and part of the work in Chapter 5.

Publication List

- P. Buchholz, T. Dayar, J. Kriege, and M. C. Orhan, Compact representation of solution vectors in Kronecker-based Markovian analysis. In G. Agha and B. V. Houdt, editors, *Proceedings of the 13th International Conference on Quantitative Evaluation of Systems* volume 9826 of *Lecture Notes in Computer Science*, pages 260–276. Springer, Heidelberg, 2016.
- T. Dayar and M. C. Orhan, On vector-Kronecker product multiplication with rectangular factors. *SIAM Journal on Scientific Computing*, 37:S526–S543, 2015.
- T. Dayar and M. C. Orhan, Cartesian product partitioning of multidimensional reachable state spaces. *Probability in the Engineering and Informational Sciences*, 30:413–430, 2016.
- T. Dayar and M. C. Orhan, Steady-state analysis of a multi-class MAP/PH/c queue with acyclic PH retrials. *Journal of Applied Probability*, 53:1098–1110, 2016.

Contents

1	Introduction	1
1.1	Kronecker Representation	3
1.2	Layout of the Thesis	5
2	Cartesian product partitioning	7
2.1	Notation and definitions	9
2.2	Cartesian product partitioning algorithms	9
2.2.1	Merge based partitioning	13
2.2.2	Refinement based partitioning	17
2.3	Experimental results	25
2.3.1	Test problems from literature	25
2.3.2	A class of random test problems	29
3	On vector–Kronecker product multiplication	34
3.1	Shuffle algorithm	36
3.2	Pot-RwCl algorithm	38
3.3	Modified shuffle algorithm	41
3.4	Implementation issues	45
3.5	Numerical results	47
4	Compact vector representation of solution vectors	54
4.1	Compact vectors in Kronecker setting	56
4.1.1	HTD format	57
4.1.2	Uniform distribution in HTD format	59
4.1.3	Multiplication of vector in HTD format with Kronecker product	59

4.1.4	Addition of two vectors in HTD format and truncation . . .	60
4.1.5	Computing the 2-norm of a vector in HTD format	61
4.2	Implementation issues	62
4.3	Numerical results	65
5	Numerical solution of the chemical master equation	71
5.1	Chemical master equation	72
5.2	Backward differentiation formulae	76
5.3	Iterative solution vectors in HTD format	78
5.3.1	Newton-Schulz method	78
5.3.2	Jacobi method	80
5.4	Implementation issues	80
5.4.1	Choosing truncated state space	81
5.4.2	Adjusting step size	84
5.4.3	Truncation strategies	84
5.5	Numerical results	86
6	A multi-class MAP/PH/c queue with acyclic PH retrials	100
6.1	Mathematical model	104
6.2	Ergodicity condition	107
6.2.1	Necessary condition	109
6.2.2	Sufficient condition	110
6.3	Numerical results	114
7	Conclusion	117

List of Figures

2.1	An arbitrary 3-dimensional orthogonal polytope $\mathcal{P}$ and its transformation $\mathcal{Q}$	11
2.2	Conflicting edges in the unit distance graph of $\mathcal{R}$ in Example 2 . .	23
2.3	Subgraphs of unit distance graph of $\mathcal{R}$ in Example 2 for Algorithm 2	24
4.1	Matrices forming $\mathbf{x}$ in HTD format for $d = 8$	57
4.2	Matrices forming $\mathbf{x}$ in HTD format for $d = 5$	58
4.3	Ranks of basis and transfer matrices forming $\boldsymbol{\pi}^{(it)}$, availability $d = 8$	70
4.4	Densities of basis and transfer matrices forming $\boldsymbol{\pi}^{(it)}$, availability $d = 8$	70
5.1	Maximum ranks of solution vectors for metabolite synthesis model	91
5.2	Maximum ranks of solution vectors for extended toggle switch model	93
5.3	Maximum ranks of solution vectors for lambda phage model . . .	95
5.4	Maximum ranks of solution vectors for cascade model with five proteins	97
5.5	Maximum ranks of solution vectors for cascade model with six molecules	99

List of Tables

2.1	Properties of models from literature	26
2.2	Experimental results for models from literature when states are processed in lexicographical order	27
2.3	Experimental results for models from literature when states are processed in random order	28
2.4	Properties of random test problems	32
2.5	Experimental results for random test problems	33
3.1	Model properties	49
3.2	Numerical results	50
3.2	Numerical results (cont'd)	51
4.1	Properties of availability and polling models	66
4.2	Numerical results for availability models	68
4.3	Numerical results for polling models	69
5.1	Transition classes of metabolite synthesis model	89
5.2	Numerical results for metabolite synthesis model	90
5.3	Transition classes of extended toggle switch model	91
5.4	Numerical results for extended toggle switch model	92
5.5	Transition classes of lambda phage model	93
5.6	Numerical results for lambda phage model	94
5.7	Transition classes of cascade model five molecules	95
5.8	Numerical results for cascade model with five molecules	96
5.9	Transition classes of cascade model with six molecules	97
5.10	Numerical results for cascade model with six molecules	98

6.1 Numerical results 116



Chapter 1

Introduction

The behavior of a physical system can be described by the states that the system may visit and how transitions from one state to another occur. The system can be modeled as a Markovian process if it exhibits the Markov property, that is, the future evolution of the system depends only on the current state of the system and not on the history of the process. A Markovian process with discrete state space is referred to as a Markov chain (MC). The Markov property requires that the time spent in a state of a MC during a visit exhibit the memoryless property, that is, time until the next transition does not depend on the time already spent in that state. The memoryless property dictates that the time spent in a state during a visit be exponentially distributed in a continuous-time MC (CTMC).

A CTMC can be represented by an $(|\mathcal{S}| \times |\mathcal{S}|)$ real matrix $\mathbf{Q}$, known as the infinitesimal generator matrix, where $\mathcal{S}$ denotes the possibly countably infinite state space of the CTMC. The off-diagonal entries of $\mathbf{Q}$ are nonnegative and indicate the rates at which transitions occur. Specifically, $\mathbf{Q}(\mathbf{x}, \mathbf{y})$ denotes the rate at which a transition occurs from state $\mathbf{x}$ to state $\mathbf{y}$ for $\mathbf{x} \neq \mathbf{y}$. The diagonal of $\mathbf{Q}$ is defined as the negated row sums of its off-diagonal entries (i.e., $\mathbf{Q}(\mathbf{x}, \mathbf{x}) = -\sum_{\mathbf{y} \neq \mathbf{x}} \mathbf{Q}(\mathbf{x}, \mathbf{y})$). Throughout our discussion, we assume that the transition rates do not depend on time, that is, the CTMC is time-homogeneous. Time-inhomogeneous CTMCs are not in the scope of this thesis.

Let $\boldsymbol{\pi}_t$ denote the transient probability distribution vector at time $t \geq 0$. By definition, $\boldsymbol{\pi}_t$ has $|\mathcal{S}|$ nonnegative entries and $\sum_{\mathbf{x}} \boldsymbol{\pi}_t(\mathbf{x}) = 1$, where $\boldsymbol{\pi}_t(\mathbf{x})$ denotes the probability of the system being in state $\mathbf{x}$ at time $t \geq 0$. The system of ordinary differential equations (ODEs) that describes the time evolution of the CTMC is given by

$$\frac{\partial \boldsymbol{\pi}_t}{\partial t} = \boldsymbol{\pi}_t \mathbf{Q} \quad \text{with initial condition} \quad \boldsymbol{\pi}_t \Big|_{t=0} = \boldsymbol{\pi}_0. \quad (1.1)$$

It follows that

$$\boldsymbol{\pi}_t = \boldsymbol{\pi}_0 e^{\mathbf{Q}t} = \boldsymbol{\pi}_0 \sum_{i=0}^{\infty} \frac{(\mathbf{Q}t)^i}{i!}.$$

There are several numerical techniques to compute the transient probability vector such as uniformization, ODE solvers, and Krylov subspace methods. When the state space is countably infinite, a CTMC obtained by truncating the state space is analyzed to approximate the transient distribution.

The limiting distribution

$$\boldsymbol{\pi} = \lim_{t \rightarrow \infty} \boldsymbol{\pi}_t$$

exists for some processes. This limit is referred to as the steady-state probability distribution and satisfies

$$\boldsymbol{\pi} \mathbf{Q} = \mathbf{0} \quad \text{and} \quad \sum_{\mathbf{x}} \boldsymbol{\pi}(\mathbf{x}) = 1.$$

Hence, $\boldsymbol{\pi}$ is a stationary distribution. The stationary distribution of a finite CTMC exists if and only if the CTMC is irreducible, i.e., each state is reachable from every other state. The existence of the stationary distribution of a countably infinite CTMC depends on the transition rates in the generator matrix as well as reachability. As for the transient analysis, the solution of the countably infinite CTMC can be computed by choosing a sufficiently large truncated state space.

Systems are usually composed of interacting subsystems. Such a system can be modeled as a multi-dimensional CTMC with each dimension of the CTMC representing a different subsystem and a number of events that trigger state changes at certain transition rates. In Kronecker-based Markovian modeling formalisms such as Stochastic Automata Networks (SANs) [129, 130], the set of rows and the

set of columns of the generator matrix are each equal to the product state space and the generator matrix is expressed as a sum of Kronecker products of subsystem transition matrices. In order to avoid unnecessary floating-point operations (flops) due to unreachable states, an auxiliary flag vector as long as the product state space can be used to indicate the reachability of states. However, this approach poses considerable inefficiency due to indexing and addressing during analysis when the reachable state space is significantly smaller than the product state space.

Compact storage of the generator matrix underlying a MC with no unreachable states and efficient implementation of analysis methods using Kronecker operations require the reachable state space to be represented as a union of Cartesian products of subsets of subsystem state spaces [43] as it has been done, for instance, in hierarchical Markovian models (HMMs) [19]. Consequently, the generator matrix can be expressed as a block matrix in which the set of rows and columns corresponding to each diagonal block are the same, each set equals to a particular partition of the reachable state space, and each block is a sum of Kronecker products of subsystem transition submatrices.

1.1 Kronecker Representation

Letting $\mathbf{A} \in \mathbb{R}^{m_A \times n_A}$ and $\mathbf{B} \in \mathbb{R}^{m_B \times n_B}$ be two rectangular matrices, the Kronecker (or tensor) product [42, 110] of $\mathbf{A}$ and $\mathbf{B}$, i.e., $\mathbf{A} \otimes \mathbf{B}$, yields the rectangular matrix $\mathbf{C} \in \mathbb{R}^{m_A m_B \times n_A n_B}$ whose entries satisfy

$$\mathbf{C}(x_C, y_C) = \mathbf{A}(x_A, y_A) \mathbf{B}(x_B, y_B)$$

$$\text{with } x_C = x_A m_B + x_B \quad \text{and} \quad y_C = y_A n_B + y_B$$

for

$$\begin{aligned} (x_A, y_A) &\in \{0, \dots, m_A - 1\} \times \{0, \dots, n_A - 1\} , \\ (x_B, y_B) &\in \{0, \dots, m_B - 1\} \times \{0, \dots, n_B - 1\} . \end{aligned}$$

Note that the Kronecker product is associative; hence, it is defined for more than two matrices.

We consider a Markovian process $\{X(t) = (X_1(t), \dots, X_d(t)) \in \times_{h=1}^d \mathcal{S}_h, t \geq 0\}$ associated with the d -dimensional CTMC, where $\mathcal{S}_h$ denotes the state space of dimension $h = 1, \dots, d$. We assume that $\mathcal{S}_h$ are defined on consecutive nonnegative integers starting from 0. Otherwise, it is always possible to enumerate the subsystem state spaces so that they satisfy this assumption. The reachable state space $\mathcal{R}$ can be partitioned into the sets $\mathcal{R}^{(1)}, \dots, \mathcal{R}^{(L)}$ (i.e., $\mathcal{R} = \cup_{l=1}^L \mathcal{R}^{(l)}$ and $\mathcal{R}^{(l)} \cap \mathcal{R}^{(l')} = \emptyset$ for $l \neq l'$), where $\mathcal{R}_h^{(l)} \subseteq \mathcal{S}_h$ consists of consecutive integers and $\mathcal{R}^{(l)} = \times_{h=1}^d \mathcal{R}_h^{(l)}$ for $l = 1, \dots, L$. Then, the infinitesimal generator $\mathbf{Q}$ can be viewed as an $(L \times L)$ block matrix induced by the partitioning $\{\mathcal{R}^{(1)}, \dots, \mathcal{R}^{(L)}\}$ as in

$$\mathbf{Q} = \begin{bmatrix} \mathbf{Q}^{(1,1)} & \dots & \mathbf{Q}^{(1,L)} \\ \vdots & \ddots & \vdots \\ \mathbf{Q}^{(L,1)} & \dots & \mathbf{Q}^{(L,L)} \end{bmatrix}.$$

Block (l, l') of $\mathbf{Q}$ for $l, l' = 1, \dots, L$ is given by

$$\mathbf{Q}^{(l,l')} = \begin{cases} \sum_{j \in \mathcal{T}^{(l,l')}} \mathbf{S}_j^{(l,l')} - \sum_{l'=1}^L \sum_{j \in \mathcal{T}^{(l,l')}} \mathbf{D}_j^{(l,l')} & \text{if } l = l', \\ \sum_{j \in \mathcal{T}^{(l,l')}} \mathbf{S}_j^{(l,l')} & \text{otherwise,} \end{cases}$$

where

$$\mathbf{S}_j^{(l,l')} = \psi_j \bigotimes_{h=1}^d \mathbf{S}_{l,h}^{(l,l')}, \quad \mathbf{D}_j^{(l,l')} = \psi_j \bigotimes_{h=1}^d \text{diag}(\mathbf{S}_{j,h}^{(l,l')} \mathbf{e}),$$

ψ_j is the rate associated with transition j , $\mathcal{T}^{(l,l')}$ is the set of transitions in block (l, l') , $\mathbf{e}$ represents a column vector of ones, and $\mathbf{S}_{j,h}^{(l,l')}$ is the submatrix of the transition matrix $\mathbf{S}_{j,h}$ whose row and column state spaces are $\mathcal{R}_h^{(l)}$ and $\mathcal{R}_h^{(l')}$, respectively. In practice, the matrices $\mathbf{S}_{j,h}$ are sparse and held in sparse row format since the nonzeros in each of its rows indicate the possible transitions from the state with that row index.

Throughout the thesis, calligraphic uppercase letters are used for sets. Vectors and matrices are respectively represented with boldface lowercase and boldface uppercase letters. All vectors are row vectors except $\mathbf{e}$ and $\mathbf{e}_h$ which represent a column vector of ones and the h th column of the identity matrix, respectively.

$\mathbf{1}$ denotes a row vector of ones and $\mathbf{I}_m$ denotes the identity matrix of order m . Lengths of vectors are determined from the contexts in which they are used. Order of an identity matrix is determined from the context if its subscript is omitted. $\text{diag}(\mathbf{a})$ denotes the diagonal matrix with the entries of vector $\mathbf{a}$ along its diagonal. $\text{nnz}(\mathbf{A})$ denotes the number of nonzeros in $\mathbf{A}$. $\mathbf{A}(\mathcal{X}, \mathcal{Y})$ is the submatrix of $\mathbf{A}$ where rows and columns are respectively selected from sets $\mathcal{X}$ and $\mathcal{Y}$, and $\mathbf{A}(\mathbf{x}, \mathbf{y})$ is the value of the matrix $\mathbf{A}$ corresponding to element $(\mathbf{x}, \mathbf{y})$. Consistently, $\mathbf{a}(\mathbf{x})$ is the value of the vector $\mathbf{a}$ at state $\mathbf{x}$ and $\mathbf{a}(\mathcal{X})$ is the subvector of $\mathbf{a}$ including the values of $\mathbf{a}(\mathbf{x})$ for $\mathbf{x} \in \mathcal{X}$. $\mathbb{Z}$, $\mathbb{Z}_{\geq 0}$, $\mathbb{R}_{\geq 0}$ and $\mathbb{R}_{> 0}$ stand for sets of integers, nonnegative integers, nonnegative real numbers, and positive real numbers, respectively. $O(\cdot)$ stands for the big O notation. The complexities of the algorithms are worst case complexities unless otherwise stated. Finally, ^{T} , $|\cdot|$, $\times$, $\otimes$, and $\star$ respectively denote transposition, the number of elements in a set, Cartesian product operator, Kronecker product operator, and elementwise multiplication operator.

1.2 Layout of the Thesis

Representing the reachable state space as a union of Cartesian products of subsets of subsystem state spaces is essential to store the generator matrix compactly and implement efficient analysis methods using Kronecker operations. The next chapter, which is based on [51], shows that the problem of partitioning the reachable state space of a three or higher dimensional system with a minimum number of partitions into Cartesian products of subsets of subsystem state spaces is NP-complete. Then it presents two algorithms that can be used to compute possibly non-optimal partitionings of a given reachable state space and investigates the performance of these algorithms through a set of problems from the literature and those that are randomly generated.

The generator matrix is not explicitly generated when it is represented compactly using Kronecker products. The implementation of efficient transient and steady-state solvers rests on efficient vector–Kronecker product multiplication

algorithms such as the shuffle algorithm. Chapter 3, which is based on [49], improves the shuffle algorithm by omitting zero rows and columns in the factors of the Kronecker product terms that result in unnecessary (flops) and shows the effectiveness of the modification through a set of problems from the literature.

The Kronecker representation of the generator matrix decreases the memory requirement; however, solution vectors used in the analysis still require to be as long as the size of the reachable state space. Chapter 4, which is based on [29], shows that it is possible to use the hierarchical Tucker decomposition (HTD) to store the solution vectors and still carry out the basic operation of vector-matrix multiplication in Kronecker form relatively efficiently.

Chapter 5 considers a stochastic chemical system modelled as a CTMC with a countably infinite multi-dimensional state space and solves the system of ODEs that describes the time evolution of the CTMC by a backward differentiation formula with vectors in HTD format. Numerical experiments indicate that adaptive truncation strategies improve time and memory efficiency significantly when fixed truncation strategies are inefficient.

Chapter 6, which is based on [53], considers a multi-class retrial queueing system with acyclic PH retrials. A necessary and sufficient condition for ergodicity is obtained from criteria based on drifts. The countably infinite state space of the model is truncated with an appropriately chosen Lyapunov function. The truncated model is described as a multi-dimensional MC and a Kronecker representation of its generator matrix is numerically analyzed.

The final chapter concludes the thesis.

Chapter 2

Cartesian product partitioning

A d -dimensional orthogonal polytope can be represented by a set of d -dimensional vectors [16]. In such a representation, the Cartesian product of sets of consecutive integers represents a hyper-rectangle. Hence, Cartesian product partitioning of a d -dimensional reachable state space is equivalent to the hyper-rectangular partitioning of the d -dimensional polytope that is represented by the reachable state space. Partitioning a 2-dimensional orthogonal polytope into a minimum number of hyper-rectangles is well studied and there are polynomial time algorithms for this problem [69, 98, 143]. However, the 3-dimensional version of this problem is shown to be NP-complete [59]. In [96], an algorithm to partition a 3-dimensional orthogonal polytope into hyper-rectangles is proposed. To the best of our knowledge, there is no algorithm to partition a d -dimensional orthogonal polytope into hyper-rectangles for $d > 3$.

Our motivation is to automate the partitioning of a given multi-dimensional reachable state space into Cartesian products of subsets of subsystem state spaces. For practical purposes, the number of partitions in the partitioning should be kept as small as possible. With this objective in mind, we first show that the problem of partitioning the reachable state space of a three or higher dimensional system with a minimum number of partitions into Cartesian products of subsets of subsystem state spaces is NP-complete [72]. Then we present two algorithms

that can be used to compute possibly non-optimal partitionings of the reachable state space into Cartesian products of subsets of subsystem state spaces.

The first algorithm starts with partitions as singletons, each representing a reachable state. Two partitions are merged if their union is also a Cartesian product of sets of consecutive integers. The partitions are merged until there are no partitions that can be merged with each other. We call this the merge based algorithm. The second algorithm takes a different approach. First, the unit distance graph of the reachable state space is constructed. The vertex set of this graph is the reachable state space and there is an edge between two vertices if the distance between them is one. Then this graph is refined [125] by removing edges until no further refinement is necessary. We call this the refinement based algorithm.

Through a set of problems from the literature [21, 26, 30] and those that are randomly generated, the performance of the two algorithms is investigated. Results indicate that although it may be more time and memory consuming, the refinement based algorithm almost always computes partitionings with a smaller number of partitions than the merge based algorithm. In many cases, the partitionings computed by the refinement based algorithm are the optimal ones. Furthermore, the refinement based algorithm is insensitive to the order in which the states in the reachable state space are processed.

The next section introduces the notation and preliminary definitions used. The second section defines the partitioning problem formally, shows that it is NP-complete, and presents two algorithms that provide, possibly non-optimal, solutions to the problem. The third section reports on experimental results with the two algorithms.

2.1 Notation and definitions

A d -dimensional hyper-rectangle is defined to be a Cartesian product of d intervals as in $\times_{h=1}^d [a_h, b_h]$, where $[a_h, b_h] \subseteq \mathbb{R}$ is an interval for $a_h < b_h$, $a_h, b_h \in \mathbb{R}$, and $h = 1, \dots, d$. A point set $\mathcal{X} \subseteq \mathbb{R}^d$ is convex if the line segment between $\mathbf{x}$ and $\mathbf{y}$ is in $\mathcal{X}$ for $\mathbf{x}, \mathbf{y} \in \mathcal{X}$. The convex hull of $\mathcal{X}$ is the smallest set containing $\mathcal{X}$. A d -dimensional convex polytope is the convex hull of a finite set $\mathcal{X} \subseteq \mathbb{R}^d$ [153]. A d -dimensional polytope is the union of a finite number of d -dimensional convex polytopes. A d -dimensional polytope is orthogonal if it is a union of d -dimensional hyper-rectangles [16].

Let $G = (\mathcal{V}, \mathcal{E})$ be an undirected graph with vertex set $\mathcal{V}$ and edge set $\mathcal{E}$. A graph $G' = (\mathcal{V}', \mathcal{E}')$ is a subgraph of $G = (\mathcal{V}, \mathcal{E})$ if $\mathcal{V}' \subseteq \mathcal{V}$ and $\mathcal{E}' \subseteq \{(\mathbf{x}, \mathbf{y}) \in \mathcal{E} \mid \mathbf{x}, \mathbf{y} \in \mathcal{V}'\}$. A subgraph of $G = (\mathcal{V}, \mathcal{E})$ induced by $\mathcal{V}' \subseteq \mathcal{V}$ is a graph whose vertex set is $\mathcal{V}'$ and edge set is $\{(\mathbf{x}, \mathbf{y}) \in \mathcal{E} \mid \mathbf{x}, \mathbf{y} \in \mathcal{V}'\}$. Two vertices of a graph are said to be connected if there exists a sequence of edges that lead from one of the vertices to the other. A subgraph of $G = (\mathcal{V}, \mathcal{E})$ forms a connected component if each pair of vertices in the subgraph are connected. A unit distance graph is a graph having a drawing in which all edges are of unit length [114].

2.2 Cartesian product partitioning algorithms

We are interested in partitionings, where the partitions themselves are Cartesian products of subsets of subsystem state spaces. Next we define the Cartesian product partitioning problem.

Definition 1. The set $\{\mathcal{R}^{(1)}, \dots, \mathcal{R}^{(L)}\}$ is said to be a Cartesian product partitioning of the multi-dimensional reachable state space $\mathcal{R}$ if $\mathcal{R}^{(l)} = \times_{h=1}^d \mathcal{R}_h^{(l)}$, the state space of the h th subsystem $\mathcal{R}_h^{(l)} \subseteq \mathcal{S}_h$ consists of consecutive integers, $\cup_{l=1}^L \mathcal{R}^{(l)} = \mathcal{R}$, and $\mathcal{R}^{(l)} \cap \mathcal{R}^{(l')} = \emptyset$ for $h = 1, \dots, d$, $l \neq l'$, $l, l' = 1, \dots, L$, and $L \in \mathbb{Z}_{>0}$.

For practical purposes that aid the use of Kronecker operations, the number of partitions in the Cartesian product partitionings of $\mathcal{R}$ should be as small as possible. Therefore, our interest lies in minimizing the number of partitions in the partitioning of $\mathcal{R}$. Unfortunately, the decision problem derived from the minimum Cartesian product partitioning problem is NP-complete [72] when the system has three or higher dimensions as we next show.

Theorem 1. It is NP-complete to decide whether there is a Cartesian product partitioning of the multi-dimensional reachable state space $\mathcal{R}$ with less than $L_{\mathcal{R}}$ partitions for given $d \in \mathbb{Z}_{>0}$, $L_{\mathcal{R}} \in \mathbb{Z}_{>0}$, and $\mathcal{R} \subseteq \mathbb{Z}_{\geq 0}^d$ when $d \geq 3$.

Proof. It is possible to check whether a set of states is a Cartesian product of sets of consecutive integers, two sets are disjoint, and the union of given sets is equal to $\mathcal{R}$ in polynomial time. Therefore, the Cartesian product partitioning problem is in class NP.

For a given arbitrary d -dimensional orthogonal polytope $\mathcal{P} \subseteq \mathbb{R}^d$ and an integer $L_{\mathcal{R}} \in \mathbb{Z}_{>0}$, it is NP-complete to decide whether $\mathcal{P}$ can be partitioned into $L_{\mathcal{R}}$ or less d -dimensional hyper-rectangles for $d \geq 3$ since it is NP-complete when instances are restricted to $d = 3$ [59]. Now, let $\mathcal{V} \subseteq \mathcal{P}$ be the vertex set of $\mathcal{P}$ and $\mathcal{V}_h = \{x_h \in \mathbb{R} \mid \mathbf{x} = (x_1, \dots, x_d) \in \mathcal{V}\}$ for $h = 1, \dots, d$. Furthermore, assume that the elements of $\mathcal{V}_h = \{v_{h,1}, \dots, v_{h,|\mathcal{V}_h|}\}$ are ordered so as to satisfy $v_{h,l} < v_{h,l'}$ if and only if $l < l'$ for $l, l' = 1, \dots, |\mathcal{V}_h|$ and $h = 1, \dots, d$.

Since $\mathcal{P}$ is an orthogonal polytope, there exists $L \in \mathbb{Z}_{>0}$ such that

$$\mathcal{P} = \bigcup_{l=1}^L \times_{h=1}^d [a_h^{(l)}, b_h^{(l)}],$$

where $\times_{h=1}^d [a_h^{(l)}, b_h^{(l)}]$ is a hyper-rectangle for $l = 1, \dots, L$. Note that $\mathcal{P}$ can be transformed to an orthogonal polytope $\mathcal{Q} \subseteq \mathbb{R}^d$ (see Figure 2.1) for which the minimum hyper-rectangular partitionings of $\mathcal{P}$ and $\mathcal{Q}$ include the same number of hyper-rectangles as we next show.

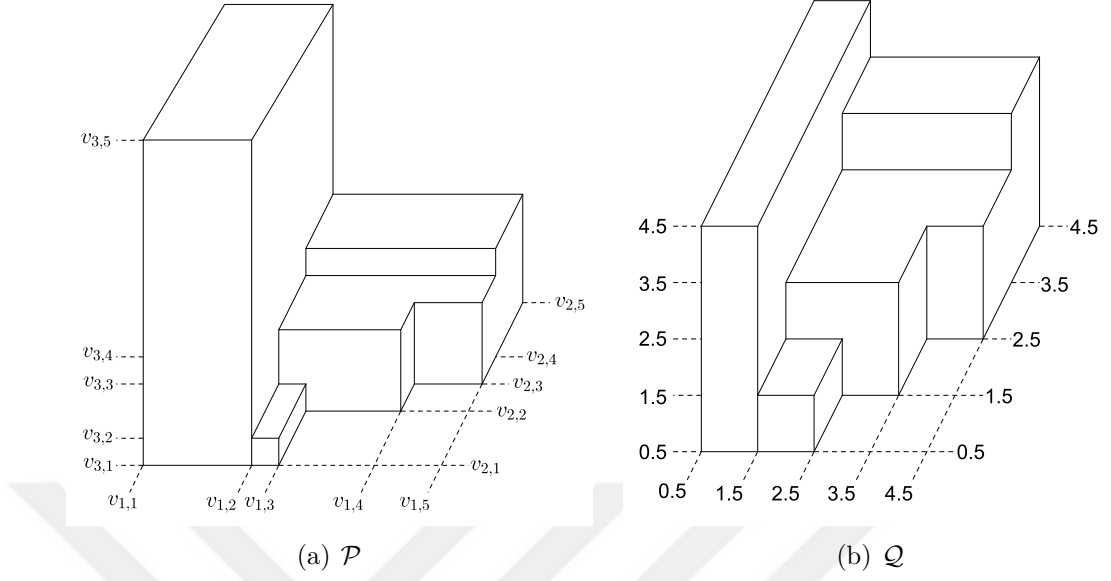


Figure 2.1: An arbitrary 3-dimensional orthogonal polytope $\mathcal{P}$ and its transformation $\mathcal{Q}$

Let

$$f_h(x_h) = \begin{cases} l - 0.5 + g_h(x_h) & \text{if } v_{h,1} \leq x_h < v_{h,|\mathcal{V}_h|} \\ |\mathcal{V}_h| - 0.5 & \text{if } x_h = v_{h,|\mathcal{V}_h|} \end{cases}$$

be the transformation function, where

$$g_h(x_h) = (x_h - v_h^{(l)}) / (v_h^{(l+1)} - v_h^{(lk)})$$

for $x_h \in [v_{h,l}, v_{h,l+1})$, $l = 1, \dots, |\mathcal{V}_h| - 1$, and $h = 1, \dots, d$. The function $f_h : [v_{h,1}, v_{h,|\mathcal{V}_h|}] \rightarrow [0.5, |\mathcal{V}_h| - 0.5]$ is continuous and increasing, so the transformation of an interval is also an interval, that is,

$$\{y_h \in \mathbb{R} \mid x_h \in [a_h, b_h], y_h = f_h(x_h)\} = [f_h(a_h), f_h(b_h)].$$

Then the set

$$\begin{aligned} \mathcal{Q} &= \{\mathbf{y} \in \mathbb{R}^d \mid \mathbf{x} \in \mathcal{P}, y_h = f_h(x_h) \text{ for } h = 1, \dots, d\} \\ &= \{\mathbf{y} \in \mathbb{R}^d \mid \mathbf{x} \in \cup_{l=1}^L \times_{h=1}^d [a_h^{(l)}, b_h^{(l)}], y_h = f_h(x_h) \text{ for } h = 1, \dots, d\} \\ &= \bigcup_{l=1}^L \times_{h=1}^d [f_h(a_h^{(l)}), f_h(b_h^{(l)})] \end{aligned}$$

is an orthogonal polytope since it is a union of hyper-rectangles, and

$$[f_h(x_h^{(l)}), f_h(b_h^{(l)})] \cap [f_h(a_h^{(l')}), f_h(b_h^{(l')})] = \emptyset$$

holds if $[a_h^{(l)}, b_h^{(l)}] \cap [a_h^{(l')}, b_h^{(l')}] = \emptyset$ for $l \neq l'$, $l, l' = 1, \dots, L$, and $h = 1, \dots, d$. Therefore, if $\mathcal{P}$ can be partitioned into L hyper-rectangles, then $\mathcal{Q}$ can be partitioned into L or less hyper-rectangles. The inverse of the function f_h is also increasing and continuous. Hence, $\mathcal{Q}$ can be transformed to $\mathcal{P}$ similarly. So, we conclude $\mathcal{Q}$ can be partitioned into L hyper-rectangles if and only if $\mathcal{P}$ can be partitioned into L hyper-rectangles.

Now, we transform the orthogonal polytope $\mathcal{Q}$ to the d -dimensional reachable state space, $\mathcal{R}$. Each vertex $\mathbf{v} = (v_1, \dots, v_d)$ of $\mathcal{Q}$ satisfies $v_h - 0.5 \in \mathbb{Z}_{\geq 0}$ for $h = 1, \dots, d$, so

$$\mathcal{Q} = \bigcup_{m=1}^M \times_{h=1}^d [x_h^{(m)} - 0.5, x_h^{(m)} + 0.5],$$

where $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(M)}\} = \mathcal{Q} \cap \mathbb{Z}_{\geq 0}^d$ for some $M \in \mathbb{Z}_{>0}$ [16].

Let $\mathcal{R} = \mathcal{Q} \cap \mathbb{Z}_{\geq 0}^d$ and $\{\mathcal{R}^{(1)}, \dots, \mathcal{R}^{(L)}\}$ be a Cartesian product partitioning of $\mathcal{R}$, where $\mathcal{R}^{(l)} = \times_{h=1}^d \mathcal{R}_h^{(l)}$ for $l = 1, \dots, L$. Then $\mathcal{Q}$ can be written as a union of disjoint hyper-rectangles, that is,

$$\begin{aligned} \mathcal{Q} &= \bigcup_{\mathbf{x} \in \mathcal{R}} \times_{h=1}^d [x_h - 0.5, x_h + 0.5] \\ &= \bigcup_{l=1}^L \times_{h=1}^d [\min(\mathcal{R}_h^{(l)}) - 0.5, \max(\mathcal{R}_h^{(l)}) + 0.5]. \end{aligned}$$

Therefore, if $\mathcal{R}$ can be partitioned into L Cartesian products of sets of consecutive integers, then $\mathcal{Q}$ can be partitioned into L or less hyper-rectangles.

Now, let $\{\mathcal{Q}^{(1)}, \dots, \mathcal{Q}^{(L)}\}$ be a hyper-rectangular partitioning of $\mathcal{Q}$, where $\mathcal{Q}^{(l)} = \times_{h=1}^d [a_h^{(l)}, b_h^{(l)}]$ for $l = 1, \dots, L$. Then

$$\mathcal{R} = \bigcup_{l=1}^L (\mathcal{Q}^{(l)} \cap \mathbb{Z}_{\geq 0}^d) = \bigcup_{l=1}^L \times_{h=1}^d ([a_h^{(l)}, b_h^{(l)}] \cap \mathbb{Z}_{\geq 0}),$$

where $[a_h^{(l)}, b_h^{(l)}] \cap \mathbb{Z}_{\geq 0}$ is a set of consecutive integers if it is not empty for $l = 1, \dots, L$ and $h = 1, \dots, d$. Besides, $\mathcal{R}^{(l)} \cap \mathcal{R}^{(l')} = \emptyset$ for $l \neq l'$ and $l, l' = 1, \dots, L$. Then $\{\mathcal{R}^{(1)}, \dots, \mathcal{R}^{(L)}\}$ is a Cartesian product partitioning of $\mathcal{R}$. Therefore if $\mathcal{Q}$ can be partitioned into L hyper-rectangles, then $\mathcal{R}$ can be partitioned into L or less Cartesian products of sets of consecutive integers. Thus, $\mathcal{R}$ can be partitioned

into L partitions if and only if $\mathcal{Q}$ can be partitioned into L hyper-rectangles. Therefore, $\mathcal{R}$ can be partitioned into less than $L_{\mathcal{R}}$ partitions if and only if $\mathcal{P}$ can be partitioned into less than $L_{\mathcal{R}}$ hyper-rectangles. Then it is NP-complete to compute the Cartesian product partitioning of $\mathcal{R} \subseteq \mathbb{Z}_{\geq 0}^d$ with less than $L_{\mathcal{R}}$ partitions. $\square$

Now, we present two algorithms to compute Cartesian product partitionings of the multi-dimensional reachable state space $\mathcal{R}$. The first algorithm starts with partitions as singletons, each representing a reachable state. A partition is merged with another partition if their union is also a Cartesian product of sets of consecutive integers. This algorithm terminates when there are no partitions that can be merged with each other. The second algorithm takes a different approach. The unit distance graph of the reachable state space is constructed. Then this graph is refined by removing edges until the vertex set of each connected component can be expressed as a Cartesian product of sets of consecutive integers.

2.2.1 Merge based partitioning

We start by defining set mergeability and then provide the condition for the mergeability of two partitions in a Cartesian product partitioning of $\mathcal{R}$.

Definition 2. Let $\mathcal{X} = \times_{h=1}^d \mathcal{X}_h$ and $\mathcal{Y} = \times_{h=1}^d \mathcal{Y}_h$ be two partitions in a Cartesian product partitioning of $\mathcal{R}$. The partitions $\mathcal{X}$ and $\mathcal{Y}$ are said to be mergeable if $\mathcal{X} \cup \mathcal{Y} = \times_{h=1}^d (\mathcal{X}_h \cup \mathcal{Y}_h)$ and $\mathcal{X}_h \cup \mathcal{Y}_h$ consists of consecutive integers for $h = 1, \dots, d$.

Lemma 1. Let $\mathcal{X} = \times_{h=1}^d \mathcal{X}_h$ and $\mathcal{Y} = \times_{h=1}^d \mathcal{Y}_h$ be two partitions in a Cartesian product partitioning of $\mathcal{R}$. The partitions $\mathcal{X}$ and $\mathcal{Y}$ are mergeable if and only if there exists some $i = 1, \dots, d$ such that $\max(\mathcal{X}_i) + 1 = \min(\mathcal{Y}_i)$ or $\max(\mathcal{Y}_i) + 1 = \min(\mathcal{X}_i)$, and $\mathcal{X}_h = \mathcal{Y}_h$ for $h = 1, \dots, i-1, i+1, \dots, d$.

Proof. ($\Leftarrow$) Let $i = 1, \dots, d$, $\max(\mathcal{X}_i) + 1 = \min(\mathcal{Y}_i)$ or $\max(\mathcal{Y}_i) + 1 = \min(\mathcal{X}_i)$,

and $\mathcal{X}_h = \mathcal{Y}_h$ for $h = 1, \dots, i-1, i+1, \dots, d$. Then

$$\begin{aligned}
\mathcal{X} \cup \mathcal{Y} &= (\times_{h=1}^d \mathcal{X}_h) \bigcup (\times_{h=1}^d \mathcal{Y}_h) \\
&= ((\times_{h=1}^{i-1} \mathcal{X}_h) \times \mathcal{X}_i \times (\times_{h=i+1}^d \mathcal{X}_h)) \\
&\quad \bigcup ((\times_{h=1}^{i-1} \mathcal{Y}_h) \times \mathcal{Y}_i \times (\times_{h=i+1}^d \mathcal{Y}_h)) \\
&= ((\times_{h=1}^{i-1} (\mathcal{X}_h \cup \mathcal{Y}_h)) \times \mathcal{X}_i \times (\times_{h=i+1}^d (\mathcal{X}_h \cup \mathcal{Y}_h))) \\
&\quad \bigcup ((\times_{h=1}^{i-1} (\mathcal{X}_h \cup \mathcal{Y}_h)) \times \mathcal{Y}_i \times (\times_{h=i+1}^d (\mathcal{X}_h \cup \mathcal{Y}_h))) \\
&= (\times_{h=1}^{i-1} (\mathcal{X}_h \cup \mathcal{Y}_h)) \times (\mathcal{X}_i \cup \mathcal{Y}_i) \times (\times_{h=i+1}^d (\mathcal{X}_h \cup \mathcal{Y}_h)) \\
&= \times_{h=1}^d (\mathcal{X}_h \cup \mathcal{Y}_h)
\end{aligned}$$

holds. The set $\mathcal{X}_h \cup \mathcal{Y}_h$ consists of consecutive elements for $h = 1, \dots, d$. Hence, $\mathcal{X}$ and $\mathcal{Y}$ are mergeable partitions.

($\Rightarrow$) Let $\mathcal{X} \subseteq \mathcal{R}$ and $\mathcal{Y} \subseteq \mathcal{R}$ be disjoint mergeable partitions. We first show that there exists $i = 1, \dots, d$ such that $\mathcal{X}_i \cap \mathcal{Y}_i = \emptyset$. Assume that $\mathcal{X}_h \cap \mathcal{Y}_h \neq \emptyset$ for $h = 1, \dots, d$. Then there exists $\mathbf{u} = (u_1, \dots, u_d) \in \mathcal{S}$ such that $u_h \in \mathcal{X}_h \cap \mathcal{Y}_h$ for $h = 1, \dots, d$. However, this contradicts the assumption that $\mathcal{X}$ and $\mathcal{Y}$ are disjoint. Thus $\mathcal{X}_i \cap \mathcal{Y}_i = \emptyset$ for some $i = 1, \dots, d$. Besides, either $\max(\mathcal{X}_i) + 1 = \min(\mathcal{Y}_i)$ or $\max(\mathcal{Y}_i) + 1 = \min(\mathcal{X}_i)$ holds since $\mathcal{X}_i \cup \mathcal{Y}_i$ consists of consecutive elements.

Now we show that $\mathcal{X}_h = \mathcal{Y}_h$ for $h = 1, \dots, i-1, i+1, \dots, d$. For the sake of contradiction, suppose there exists $j = 1, \dots, i-1, i+1, \dots, d$ such that $\mathcal{X}_j \neq \mathcal{Y}_j$ (i.e., $(\mathcal{X}_j \setminus \mathcal{Y}_j) \cup (\mathcal{Y}_j \setminus \mathcal{X}_j) \neq \emptyset$). Without loss of generality, assume that $\mathcal{X}_j \setminus \mathcal{Y}_j \neq \emptyset$. Then there exists $\mathbf{u} = (u_1, \dots, u_d) \in \times_{h=1}^d (\mathcal{X}_h \cup \mathcal{Y}_h)$, where $u_i \in \mathcal{Y}_i \setminus \mathcal{X}_i$ and $u_j \in \mathcal{X}_j \setminus \mathcal{Y}_j$. Therefore, $\mathbf{u} \notin \mathcal{X} \cup \mathcal{Y}$ since $u_i \notin \mathcal{X}_i$ and $u_j \notin \mathcal{Y}_j$. Then $\mathcal{X} \cup \mathcal{Y} \neq \times_{h=1}^d (\mathcal{X}_h \cup \mathcal{Y}_h)$ holds, but this contradicts the assumption that $\mathcal{X}$ and $\mathcal{Y}$ are mergeable partitions. Therefore $\mathcal{X}_h = \mathcal{Y}_h$ for $h = 1, \dots, i-1, i+1, \dots, d$. $\square$

Now, we demonstrate the concept of mergeability on an example.

Example 1. Let $d = 3$, $\mathcal{S}_h = \{0, 1, 2\}$ for $h = 1, 2, 3$, and let $\mathcal{X} = \{0, 1\} \times \{1\} \times \{2\}$ be a partition in a Cartesian product partitioning of some $\mathcal{R} \subseteq \times_{h=1}^3 \mathcal{S}_h$. Then by Lemma 1, the partitions that can be merged with $\mathcal{X}$ are $\{2\} \times \{1\} \times \{2\}$, $\{0, 1\} \times \{0\} \times \{2\}$, $\{0, 1\} \times \{2\} \times \{2\}$, $\{0, 1\} \times \{1\} \times \{1\}$, and $\{0, 1\} \times \{1\} \times \{0, 1\}$.

Let $\mathcal{X} = \times_{h=1}^d \mathcal{X}_h$ be a partition in a Cartesian product partitioning of $\mathcal{R}$. The states $(\min(\mathcal{X}_1), \dots, \min(\mathcal{X}_d)) \in \mathcal{X}$ and $(\max(\mathcal{X}_1), \dots, \max(\mathcal{X}_d)) \in \mathcal{X}$ are said to be the end states of the partition $\mathcal{X}$. Due to Lemma 1, mergeability of two partitions can be determined by their end states. Besides, the states in a partition can be obtained from its end states. Therefore, it is sufficient to keep only the end states of the partitions instead of maintaining the partitions by using a disjoint-set data structure and a union-find algorithm [41] to merge two partitions.

Input: d -dimensional reachable state space: $\mathcal{R}$
Output: Cartesian product partitioning of $\mathcal{R}$: $\mathcal{Q}$

```

1: function MERGEBASEDPARTITIONING( $\mathcal{R}, \mathcal{Q}$ )
2:    $\mathcal{Q} \leftarrow \emptyset$ 
3:   for all  $\mathbf{x} \in \mathcal{R}$  do
4:      $\mathcal{X} \leftarrow \{\mathbf{x}\}$ 
5:     while there exists some  $\mathcal{Y} \in \mathcal{Q}$  mergeable with  $\mathcal{X}$  do
6:        $\mathcal{X} \leftarrow \mathcal{X} \cup \mathcal{Y}; \mathcal{Q} \leftarrow \mathcal{Q} \setminus \{\mathcal{Y}\}$ 
7:     end while
8:      $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{\mathcal{X}\}$ 
9:   end for
10: end function

```

Algorithm 1: Merge based algorithm to compute a Cartesian product partitioning of given multi-dimensional reachable state space

In Algorithm 1, we provide the merge based Cartesian product partitioning of $\mathcal{R}$. It starts by constructing a singleton for each state in $\mathcal{R}$. For each partition that is a singleton or is obtained by merging two partitions, a mergeable partition is sought. If such a partition is located, the two partitions are merged. There are $|\mathcal{R}|$ partitions; hence, the total time complexity of constructing a new partition is $O(d|\mathcal{R}|)$. A new partition is obtained when a singleton is constructed or two partitions are merged. Initially there are $|\mathcal{R}|$ partitions, and the partitions in the partitioning never get split, implying there can be at most $(|\mathcal{R}| - 1)$ merge operations. Then the total cost of merging two partitions is $O(d|\mathcal{R}|)$. Since each partition needs to check $O(d)$ end states for mergeability with another partition, a mergeable partition is sought $O(d|\mathcal{R}|)$ times.

The efficiency of the algorithm depends on the data structure used to keep the end states of the partitions. When a balanced tree such as an AVL tree [4] is used to keep the end states, the cost of searching for a partition becomes $O(\lg(L_{max}))$

time, where L_{max} is the maximum number of partitions during the execution of the algorithm. Therefore, the time complexity of the algorithm associated with seeking a mergeable partition is $O(d|\mathcal{R}|\lg(L_{max}))$ when the end states of the partitions are kept in a balanced tree. Another cost in the algorithm is inserting end states to the tree when no mergeable partition is found and removing the end states from the tree when two partitions are merged. The time complexities of these operations are also $O(|\mathcal{R}|\lg(L_{max}))$. Therefore, the time complexity of the algorithm is $O(d|\mathcal{R}|\lg(L_{max}))$. The space requirement for each partition is $O(d)$; hence, the space requirement of the algorithm is $O(dL_{max})$. In the worst case, no partitions are merged and $L_{max} = |\mathcal{R}|$; hence, the time and space complexities of the algorithm are $O(d|\mathcal{R}|\lg(|\mathcal{R}|))$ and $O(d|\mathcal{R}|)$, respectively. If L_{max} is constant, the number of partitions is bounded by a constant during the execution of the algorithm. In that case, the time and space complexities of the algorithm become $O(d|\mathcal{R}|)$ and $O(d)$, respectively.

Observe that the partition computed by Algorithm 1 depends on the order in which the states of $\mathcal{R}$ are processed. Now, let us consider the following 3-dimensional example.

Example 2. Let $\mathcal{S}_h = \{0, 1, 2, 3\}$ for $h = 1, 2, 3$,

$$\begin{aligned} \mathcal{R} = & \{(0, 0, 1), (0, 1, 0), (0, 1, 1), (1, 0, 0), (1, 0, 1), (1, 1, 0), (1, 1, 1), (2, 0, 0), \\ & (2, 0, 1), (2, 1, 0), (2, 1, 1), (3, 0, 1), (3, 1, 0)\}, \end{aligned}$$

and assume that the states in $\mathcal{R}$ are processed in lexicographical order.

When the first 8 states are processed, the partitions are $\{(0, 0, 1), (0, 1, 1)\}$, $\{(1, 1, 1)\}$, $\{(0, 1, 0), (1, 1, 0)\}$, $\{(1, 0, 0), (1, 0, 1)\}$, and $\{(2, 0, 0)\}$. Then the singleton $\{(2, 0, 1)\}$ is merged with the singleton $\{(2, 0, 0)\}$, and their union is merged with $\{(1, 0, 0), (1, 0, 1)\}$. Finally, the remaining states are processed.

The number of partitions in the Cartesian product partitioning of $\mathcal{R}$ turns out to be 5, and the partitions are given by $\{(0, 1, 0), (1, 1, 0), (2, 1, 0), (3, 1, 0)\}$, $\{(1, 1, 1), (2, 1, 1)\}$, $\{(0, 0, 1), (0, 1, 1)\}$, $\{(1, 0, 0), (1, 0, 1), (2, 0, 0), (2, 0, 1)\}$, and $\{(3, 0, 1)\}$.

2.2.2 Refinement based partitioning

The refinement based partitioning algorithm constructs the unit distance graph of the multi-dimensional reachable state space $\mathcal{R}$. The vertex set of this graph is $\mathcal{R}$ and there is an edge between two vertices $\mathbf{x}, \mathbf{y} \in \mathcal{R}$ if $\mathbf{x} - \mathbf{y} \in \cup_{h=1}^d \{-\mathbf{e}_h, \mathbf{e}_h\}$. In other words, two vertices are adjacent in the unit distance graph if there is consecutiveness of the vertices along a particular dimension while the values of states variables in other dimensions remain constant. We next define conflicting edges in a subgraph of the unit distance graph.

Definition 3. Let $G = (\mathcal{R}, \mathcal{E})$ be a subgraph of the unit distance graph of $\mathcal{R}$ and let $\mathbf{x}, \mathbf{x} + \boldsymbol{\delta}_i, \mathbf{x} + \boldsymbol{\delta}_j \in \mathcal{R}$ for some $i, j = 1, \dots, d$, $i \neq j$, $\boldsymbol{\delta}_i \in \{-\mathbf{e}_i, \mathbf{e}_i\}$, and $\boldsymbol{\delta}_j \in \{-\mathbf{e}_j, \mathbf{e}_j\}$. Two edges $(\mathbf{x}, \mathbf{x} + \boldsymbol{\delta}_i), (\mathbf{x}, \mathbf{x} + \boldsymbol{\delta}_j) \in \mathcal{E}$ are said to be conflicting if $\mathbf{x} + \boldsymbol{\delta}_i + \boldsymbol{\delta}_j \notin \mathcal{R}$ or $\{(\mathbf{x} + \boldsymbol{\delta}_i, \mathbf{x} + \boldsymbol{\delta}_i + \boldsymbol{\delta}_j), (\mathbf{x} + \boldsymbol{\delta}_j, \mathbf{x} + \boldsymbol{\delta}_i + \boldsymbol{\delta}_j)\} \not\subseteq \mathcal{E}$.

The next lemma and corollary show that in a subgraph of the unit distance graph of $\mathcal{R}$ with no conflicting edges, the vertices in each connected component can be written as a Cartesian product of sets of consecutive integers. Therefore, eliminating conflicting edges in the unit distance graph of $\mathcal{R}$ leads to a Cartesian product partitioning of $\mathcal{R}$.

Lemma 2. Let $G = (\mathcal{R}, \mathcal{E})$ be a subgraph of the unit distance graph of $\mathcal{R}$ with no conflicting edges, $M \in \mathbb{Z}_{>0}$, $\mathbf{x}^{(m)} = (x_1^{(m)}, \dots, x_d^{(m)}) \in \mathcal{R}$ for $m = 0, \dots, M$, and

$$\mathcal{X}(M) = \times_{h=1}^d \left\{ \min_{m=0}^M (x_h^{(m)}), \dots, \max_{m=0}^M (x_h^{(m)}) \right\}.$$

If $(\mathbf{x}^{(m-1)}, \mathbf{x}^{(m)}) \in \mathcal{E}$ for $m = 1, \dots, M$, then $\mathcal{X}(M) \subseteq \mathcal{R}$ and $(\mathbf{u}, \mathbf{v}) \in \mathcal{E}$ for $\mathbf{u} - \mathbf{v} \in \cup_{h=1}^d \{-\mathbf{e}_h, \mathbf{e}_h\}$ and $\mathbf{u}, \mathbf{v} \in \mathcal{X}(M)$.

Proof. The proof is by mathematical induction on m . The base case is trivial. Assume that the statement holds when $(\mathbf{x}^{(m-1)}, \mathbf{x}^{(m)}) \in \mathcal{E}$ for $m = 1, \dots, M-1$. Now, let $(\mathbf{x}^{(M-1)}, \mathbf{x}^{(M)}) \in \mathcal{E}$. If $\mathcal{X}(M) \setminus \mathcal{X}(M-1) = \emptyset$, the statement holds by

the inductive hypothesis. Otherwise,

$$\begin{aligned}\mathcal{X}(M) \setminus \mathcal{X}(M-1) &= \left(\times_{h=1}^{i-1} \left\{ \min_{m=1}^{M-1}(x_h^{(m)}), \dots, \max_{m=1}^{M-1}(x_h^{(m)}) \right\} \right) \times \{x_i^{(M-1)} + \delta\} \\ &\quad \times \left(\times_{h=i+1}^d \left\{ \min_{m=1}^{M-1}(x_h^{(m)}), \dots, \max_{m=1}^{M-1}(x_h^{(m)}) \right\} \right)\end{aligned}$$

is not empty, where $\delta = x_i^{(M)} - x_i^{(M-1)} \in \{-1, 1\}$ for some $i = 1, \dots, d$. Let

$$\mathcal{Y} = \{\mathbf{y} \in \mathcal{X}(M) \setminus \mathcal{X}(M-1) \mid \mathbf{y} \notin \mathcal{R} \text{ or } (\mathbf{y}, \mathbf{y} - \delta \mathbf{e}_i) \notin \mathcal{E}\}$$

and assume that $\mathcal{Y} \neq \emptyset$. Then there exists $\mathbf{x}, \mathbf{y} \in \mathcal{X}(M) \setminus \mathcal{X}(M-1)$ such that $\mathbf{x} \notin \mathcal{Y}$, $\mathbf{y} \in \mathcal{Y}$, and $\mathbf{x} - \mathbf{y} \in \cup_{h=1}^d \{-\mathbf{e}_h, \mathbf{e}_h\}$. In this case, $(\mathbf{x} - \delta \mathbf{e}_i, \mathbf{y} - \delta \mathbf{e}_i) \in \mathcal{E}$ by the inductive hypothesis and $(\mathbf{x}, \mathbf{x} - \delta \mathbf{e}_i) \in \mathcal{E}$ since $\mathbf{x} \notin \mathcal{Y}$. Then $(\mathbf{x}, \mathbf{x} - \delta \mathbf{e}_i)$ and $(\mathbf{x} - \delta \mathbf{e}_i, \mathbf{y} - \delta \mathbf{e}_i)$ are conflicting edges in G , but this contradicts the assumption that G does not contain conflicting edges. Hence, $\mathcal{Y} = \emptyset$, that is, $\mathbf{u} \in \mathcal{R}$ and $(\mathbf{u}, \mathbf{u} - \delta \mathbf{e}_i) \in \mathcal{E}$ for $\mathbf{u} \in \mathcal{X}(M) \setminus \mathcal{X}(M-1)$. Now, let $\mathbf{x}, \mathbf{y} \in \mathcal{X}(M) \setminus \mathcal{X}(M-1)$ be two arbitrary vertices such that $\mathbf{x} - \mathbf{y} \in \cup_{h=1}^d \{-\mathbf{e}_h, \mathbf{e}_h\}$. Since $\mathcal{Y} = \emptyset$, $\{\mathbf{x}, \mathbf{y}\} \subseteq \mathcal{R}$ and

$$\{(\mathbf{x}, \mathbf{x} - \delta \mathbf{e}_i), (\mathbf{y}, \mathbf{y} - \delta \mathbf{e}_i), (\mathbf{x} - \delta \mathbf{e}_i, \mathbf{y} - \delta \mathbf{e}_i)\} \subseteq \mathcal{E}.$$

Then $(\mathbf{x}, \mathbf{y}) \in \mathcal{E}$ since there are no conflicting edges in G . Therefore, the statement also holds for M . $\square$

Corollary 1. Let $G = (\mathcal{R}, \mathcal{E})$ be a subgraph of the unit distance graph of $\mathcal{R}$ with no conflicting edges and $G' = (\mathcal{R}', \mathcal{E}')$ be a connected component in G . Then $\mathcal{R}'$ can be written as a Cartesian product of consecutive integers, that is, $\mathcal{R}' = \times_{h=1}^d \{u_h, \dots, v_h\}$, where $\mathbf{u} = (u_1, \dots, u_d) \in \mathcal{R}$, $\mathbf{v} = (v_1, \dots, v_d) \in \mathcal{R}$, and $u_h \leq v_h$ for $h = 1, \dots, d$.

The next definition introduces the concept of a separator, which is used in refining the unit distance graph of $\mathcal{R}$.

Definition 4. Let $\mathcal{U}, \mathcal{V} \subseteq \mathcal{R}$ and $G = (\mathcal{R}, \mathcal{E})$ be a subgraph of the unit distance graph of $\mathcal{R}$. The set of edges $\mathcal{Z} \subseteq \mathcal{E}$ is said to be a separator if

1. each edge in $\mathcal{Z}$ is incident to a vertex in $\mathcal{U}$ and a vertex in $\mathcal{V}$,

2. subgraphs of G induced by the sets $\mathcal{U}$ and $\mathcal{V}$ are each connected,
3. there does not exist two vertices $\mathbf{x}, \mathbf{y} \in \mathcal{R}$ such that $(\mathbf{x}, \mathbf{x} + \delta \mathbf{e}_h) \in \mathcal{Z}$, $(\mathbf{y}, \mathbf{y} + \delta \mathbf{e}_h) \in \mathcal{E} \setminus \mathcal{Z}$, $\{(\mathbf{x}, \mathbf{y}), (\mathbf{x} + \delta \mathbf{e}_h, \mathbf{y} + \delta \mathbf{e}_h)\} \subseteq \mathcal{E}$, and
4. there exists at least one edge in $\mathcal{Z}$ that conflicts with some edge in $\mathcal{E}$,

where $\mathcal{U} \subseteq \{\mathbf{x} \in \mathcal{R} \mid x_h = i\}$ and $\mathcal{V} \subseteq \{\mathbf{x} \in \mathcal{R} \mid x_h = i + \delta\}$ for some $i \in \mathcal{S}_h$, $\delta \in \{-1, 1\}$, and $h = 1, \dots, d$.

The next lemma shows that removing the edges in a separator decreases the number of conflicting edges in a subgraph of the unit distance graph of $\mathcal{R}$.

Lemma 3. Let $G = (\mathcal{R}, \mathcal{E})$ be a subgraph of the unit distance graph of $\mathcal{R}$ and $\mathcal{Z}$ be a separator in G . Two edges in $G' = (\mathcal{R}, \mathcal{E} \setminus \mathcal{Z})$ conflict only if they also conflict in G .

Proof. Let $(\mathbf{x}, \mathbf{x} + \boldsymbol{\delta}_i)$ and $(\mathbf{x}, \mathbf{x} + \boldsymbol{\delta}_j)$ be two edges that do not conflict in G for some $i, j = 1, \dots, d$, $i \neq j$, $\boldsymbol{\delta}_i \in \{-\mathbf{e}_i, \mathbf{e}_i\}$, and $\boldsymbol{\delta}_j \in \{-\mathbf{e}_j, \mathbf{e}_j\}$. Then $\mathbf{x} + \boldsymbol{\delta}_i + \boldsymbol{\delta}_j \in \mathcal{R}$ and $\{(\mathbf{x} + \boldsymbol{\delta}_i, \mathbf{x} + \boldsymbol{\delta}_i + \boldsymbol{\delta}_j), (\mathbf{x} + \boldsymbol{\delta}_j, \mathbf{x} + \boldsymbol{\delta}_i + \boldsymbol{\delta}_j)\} \subseteq \mathcal{E}$. Due to Definition 4, $(\mathbf{x}, \mathbf{x} + \boldsymbol{\delta}_i) \in \mathcal{Z}$ if and only if $(\mathbf{x} + \boldsymbol{\delta}_j, \mathbf{x} + \boldsymbol{\delta}_i + \boldsymbol{\delta}_j) \in \mathcal{Z}$. Similarly, $(\mathbf{x}, \mathbf{x} + \boldsymbol{\delta}_j) \in \mathcal{Z}$ if and only if $(\mathbf{x} + \boldsymbol{\delta}_i, \mathbf{x} + \boldsymbol{\delta}_i + \boldsymbol{\delta}_j) \in \mathcal{Z}$. Then,

$$\{(\mathbf{x} + \boldsymbol{\delta}_i, \mathbf{x} + \boldsymbol{\delta}_i + \boldsymbol{\delta}_j), (\mathbf{x} + \boldsymbol{\delta}_j, \mathbf{x} + \boldsymbol{\delta}_i + \boldsymbol{\delta}_j)\} \subseteq \mathcal{E} \setminus \mathcal{Z}$$

if $\{(\mathbf{x}, \mathbf{x} + \boldsymbol{\delta}_i), (\mathbf{x}, \mathbf{x} + \boldsymbol{\delta}_j)\} \subseteq \mathcal{E} \setminus \mathcal{Z}$. Therefore, if the two edges in G do not conflict, then they do not conflict in G' . $\square$

Now, we are in a position to provide the refinement based algorithm for Cartesian product partitioning of $\mathcal{R}$ (see Algorithm 2). At the outset, the unit distance graph of $\mathcal{R}$ is constructed. Then the separators in this graph are constructed and inserted to a priority queue [41], where the priority of a separator is the total number of edges which conflict with an edge in the separator. The graph is refined by removing the separator with maximum priority until no conflicting edges remain. The edge set of the graph changes when a separator is removed; hence,

the separators need to be reconstructed. By Lemma 3, an edge is conflicting in the refined graph only if it is also conflicting before refinement. Hence, each separator in the refined graph is a subset of a separator in the graph before refinement. Therefore, in order to reconstruct the separators, it is necessary and sufficient to visit the vertices incident to the edges in the separators intersecting the removed separator, where two separators are said to be intersecting if they include edges along different dimensions incident to the same vertex. If all conflicting edges are eliminated, then the vertices in each connected component can be written as a union of Cartesian products of sets of consecutive integers by Corollary 1.

Input: d-dimensional reachable state space: $\mathcal{R}$ Output: Cartesian product partitioning of $\mathcal{R}$: $\mathcal{Q}$ <pre> 1: function REFINEMENTBASEDPARTITIONING($\mathcal{R}, \mathcal{Q}$) 2: Construct unit distance graph of $\mathcal{R}$, $G = (\mathcal{R}, \mathcal{E})$ 3: Construct an empty priority queue $\mathcal{PQ}$ 4: for all $\mathbf{x} \in \mathcal{R}$ do 5: for all $h = 1, \dots, d$ do 6: if $\mathbf{x} + \mathbf{e}_h \in \mathcal{R}$ then 7: if $(\mathbf{x}, \mathbf{x} + \mathbf{e}_h)$ conflicts with some edge in $\mathcal{E}$ 8: AND $(\mathbf{x}, \mathbf{x} + \mathbf{e}_h)$ is not in a separator then 9: Construct separator $\mathcal{Z}$ including $(\mathbf{x}, \mathbf{x} + \mathbf{e}_h)$; Insert $\mathcal{Z}$ to $\mathcal{PQ}$ 10: end if 11: end if 12: end for 13: end for </pre>
--

Algorithm 2: Refinement based algorithm to compute a Cartesian product partitioning of given multi-dimensional reachable state space

In order to construct the unit distance graph, first all adjacent vertices need to be determined. In order to facilitate this, each vertex keeps an adjacency list of length $2d$ to identify the adjacent vertices. The vertices are first inserted to an AVL tree [4]. Then for each vertex, vertices that might be adjacent in the product state space are sought in the tree. There are $2d$ such vertices; hence, $2d$ vertices are sought for each vertex. Therefore, the space complexity of maintaining the graph becomes $O(d|\mathcal{R}|)$ and the time complexity of constructing the unit distance graph becomes $O(d|\mathcal{R}| \lg(|\mathcal{R}|))$. After the graph is constructed, the AVL tree is destroyed.

After the adjacency list of each vertex is set, separators need to be constructed.

```

14:  while  $\mathcal{PQ}$  is not empty do
15:     $\mathcal{Z}_{max} \leftarrow$  separator  $\mathcal{Z}$  in  $\mathcal{PQ}$  whose priority is maximum
16:    Remove  $\mathcal{Z}_{max}$  from  $\mathcal{PQ}$ 
17:    for all  $(\mathbf{x}, \mathbf{y}) \in \mathcal{Z}_{max}$  do
18:       $\mathcal{L} \leftarrow \emptyset$ 
19:      for all separators  $\mathcal{Z}$  in  $\mathcal{PQ}$  including an edge incident to  $\mathbf{x}$  or  $\mathbf{y}$  do
20:         $\mathcal{L} \leftarrow \mathcal{L} \cup \{\mathcal{Z}\}$ ; Remove  $\mathcal{Z}$  from  $\mathcal{PQ}$ 
21:      end for
22:    end for
23:     $\mathcal{E} \leftarrow \mathcal{E} \setminus \mathcal{Z}_{max}$ 
24:    for all  $\mathcal{Z} \in \mathcal{L}$  do
25:      while  $\mathcal{Z} \neq \emptyset$  do
26:        Construct the separator  $\mathcal{Z}'$  including some edge  $(\mathbf{x}, \mathbf{y}) \in \mathcal{Z}$ ;  $\mathcal{Z} \leftarrow \mathcal{Z} \setminus \mathcal{Z}'$ 
27:        if priority of  $\mathcal{Z}'$  is positive then
28:          Insert  $\mathcal{Z}'$  to  $\mathcal{PQ}$ 
29:        end if
30:      end while
31:    end for
32:  end while
33:   $\mathcal{Q} \leftarrow \emptyset$ 
34:  for all connected components  $G' = (\mathcal{R}', \mathcal{E}')$  of  $G = (\mathcal{R}, \mathcal{E})$  do
35:     $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{\mathcal{R}'\}$ 
36:  end for
37: end function

```

Algorithm 2: Refinement based algorithm to compute a Cartesian product partitioning of given multi-dimensional reachable state space (cont'd)

Each separator includes at least one conflicting edge, that is, its priority is positive. Algorithm 2 visits each edge and checks whether it conflicts with some other edge. A separator is formed by the two vertex sets ($\mathcal{U}$ and $\mathcal{V}$ in Definition 4) and the subgraphs induced by these two sets are connected. Besides, the separator is maximal; hence, the separator including a particular edge is unique and only one separator can include the edge (see Item 3 of Definition 4). Therefore, the separator including an edge is constructed only once in the unit distance graph. While constructing a separator $\mathcal{Z}$ including the edge $(\mathbf{x}, \mathbf{y})$, a breadth-first search starting at $\mathbf{x}$ is used to visit the vertices connected to $\mathbf{x}$ [41]. The time complexity of constructing the separator $\mathcal{Z}$ is $O(d|\mathcal{Z}|)$. Since each edge is added to at most one separator, the number of edges in the union of separators is $O(|\mathcal{E}|)$. Therefore, the total time complexity of constructing all separators in the unit distance graph becomes $O(d|\mathcal{E}|)$. At each vertex, we use an array of size $2d$ to keep the separators including the edge incident to that vertex. The space complexity of the algorithm remains as $O(d|\mathcal{R}|)$, but it takes constant time to retrieve the separator

including a given edge.

When a separator is constructed, it is added to the priority queue. We use a binary heap as the priority queue. Observe that the number of separators does not exceed $|\mathcal{E}|$ since each separator includes at least one edge. Therefore, the cost of inserting separators to the priority queue is $O(|\mathcal{E}| \lg(|\mathcal{E}|))$. We consider an array implementation of the priority queue and the maximum number of separators is not known in advance; therefore, we allocate $O(|\mathcal{E}|) = O(d|\mathcal{R}|)$ space for the priority queue.

After the separators in the unit distance graph are inserted to the priority queue, the graph is refined by removing the edges in the separator with maximum priority until no separators remain in the priority queue. The graph is refined at most $|\mathcal{E}|$ times since at least one edge is removed from the graph at each refinement step. At each refinement step, the separator with maximum priority, $\mathcal{Z}_{max}$, is chosen. When $\mathcal{Z}_{max}$ is removed, the intersecting separators need to be reconstructed. Hence, at each refinement step, all edges of $\mathcal{Z}_{max}$ are visited to obtain the intersecting separators. These separators are inserted to a list $\mathcal{L}$ and removed from the heap. Then these separators are reconstructed for the edge set $\mathcal{E} \setminus \mathcal{Z}_{max}$ and inserted to the heap. Reconstruction of a separator $\mathcal{Z}$ requires visiting all the vertices incident to the edges in it. The number of edges in the union of separators intersecting with $\mathcal{Z}_{max}$ is $O(d|\mathcal{E}|)$; hence, reconstruction of separators at each step has time complexity $O(d|\mathcal{E}|)$. Therefore, the total time complexity of reconstructing separators becomes $O(d|\mathcal{E}|^2)$ since reconstruction is required in all refinement steps. Now, consider the costs related to the heap operations. At each refinement step, the separators intersecting with $\mathcal{Z}_{max}$ are removed from the heap. The separator $\mathcal{Z}_{max}$ intersects with $O(d|\mathcal{Z}_{max}|)$ separators. Therefore, during the execution of the algorithm, the total number of removals from the heap is $O(d|\mathcal{E}|)$ since the total size of the separators removed from the graph is $O(|\mathcal{E}|)$. The algorithm starts and ends with an empty heap, so the number of insertions to the heap is also $O(d|\mathcal{E}|)$. Hence, the total complexity of heap operations is $O(d|\mathcal{E}| \lg(d|\mathcal{E}|))$. When all costs are considered, the time and space complexities of Algorithm 2 are $O(d|\mathcal{E}|^2) = O(d^3|\mathcal{R}|^2)$ and $O(d|\mathcal{R}|)$, respectively.

Example 2. (cont'd) Algorithm 2 first constructs the unit distance graph of $\mathcal{R}$ and then refines the graph by removing edges. Let $G^{(j)} = (\mathcal{R}, \mathcal{E}^{(j)})$ denote the subgraph of the unit distance graph of $\mathcal{R}$, where $\mathcal{E}^{(j)}$ is the edge set after j separators are removed. In the unit distance graph of $\mathcal{R}$, $G^{(0)}$, there are 7 pairs of conflicting edges (see Figure 2.2).

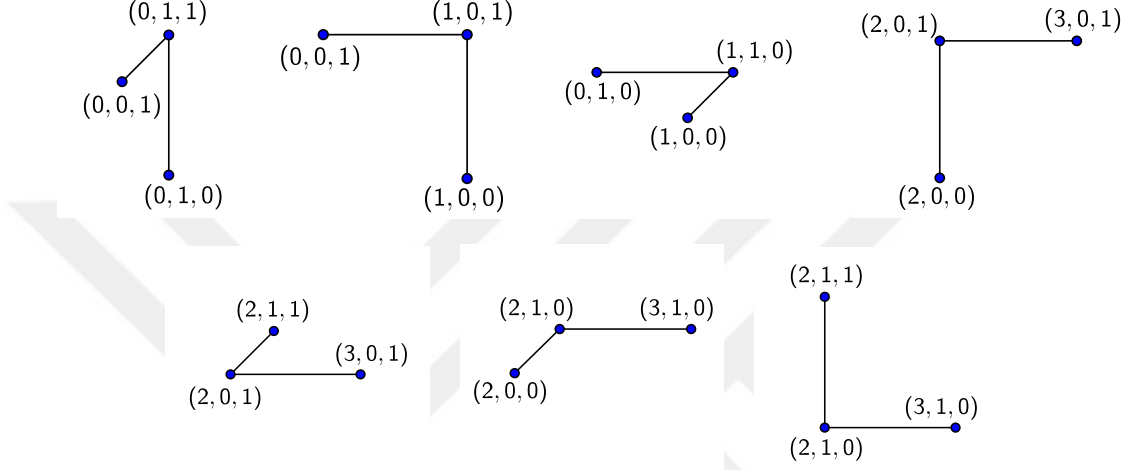


Figure 2.2: Conflicting edges in the unit distance graph of $\mathcal{R}$ in Example 2

The separators in the unit distance graph of $\mathcal{R}$ (see Figure 2.3(a)) are

$$\begin{aligned}
\mathcal{Z}^{(0,1)} &= \{((0,0,1), (0,1,1)), ((1,0,0), (1,1,0)), ((1,0,1), (1,1,1)), \\
&\quad ((2,0,0), (2,1,0)), ((2,0,1), (2,1,1))\} \\
\mathcal{Z}^{(0,2)} &= \{((0,1,0), (0,1,1)), ((1,0,0), (1,0,1)), ((1,1,0), (1,1,1)), \\
&\quad ((2,0,0), (2,0,1)), ((2,1,0), (2,1,1))\} \\
\mathcal{Z}^{(0,3)} &= \{((0,0,1), (1,0,1)), ((0,1,0), (1,1,0)), ((0,1,1), (1,1,1))\} \\
\mathcal{Z}^{(0,4)} &= \{((2,0,1), (3,0,1))\}, \mathcal{Z}^{(0,5)} = \{((2,1,0), (3,1,0))\}.
\end{aligned}$$

The priorities of the separators $\mathcal{Z}^{(0,1)}$ and $\mathcal{Z}^{(0,2)}$ are 4, and the priorities of the other separators are 2. There are two separators with maximum priority and one of them needs to be chosen. Let the edges in $\mathcal{Z}^{(0,1)}$ be chosen for removal. Then all other separators are reconstructed and 3 conflicting edges remain in $G^{(1)}$ (see

Figure 2.3(b)). The separators in this graph are

$$\begin{aligned}\mathcal{Z}^{(1,1)} &= \{((1, 0, 0), (1, 0, 1)), ((2, 0, 0), (2, 0, 1))\}, \\ \mathcal{Z}^{(1,2)} &= \{((0, 1, 0), (0, 1, 1)), ((1, 1, 0), (1, 1, 1)), ((2, 1, 0), (2, 1, 1))\}, \\ \mathcal{Z}^{(1,3)} &= \{((0, 0, 1), (1, 0, 1))\}, \quad \mathcal{Z}^{(1,4)} = \mathcal{Z}^{(0,4)}, \quad \mathcal{Z}^{(1,5)} = \mathcal{Z}^{(0,5)}.\end{aligned}$$

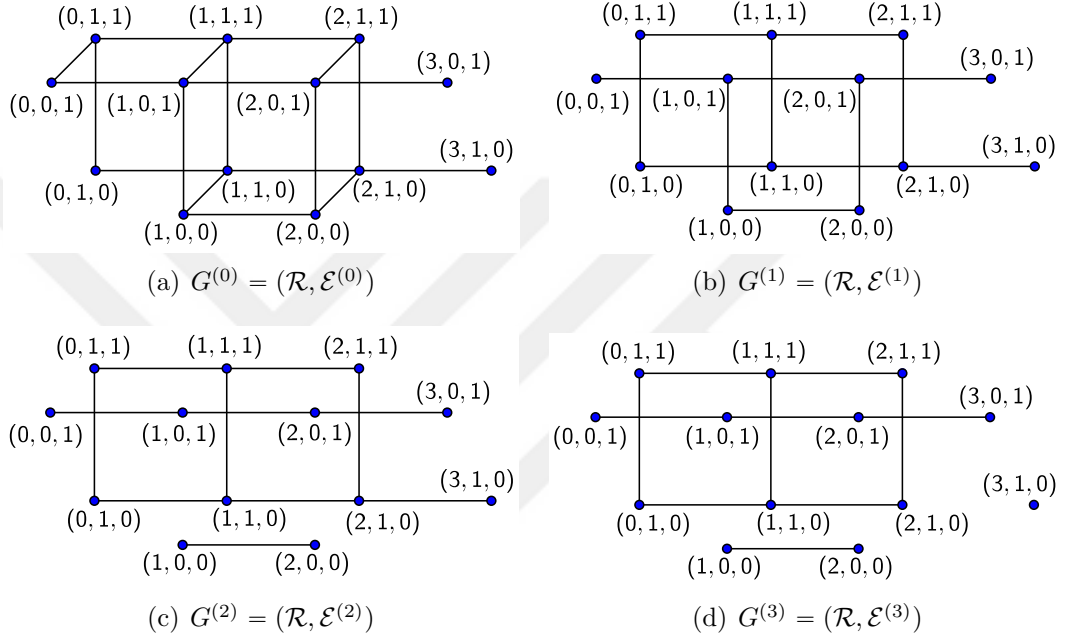


Figure 2.3: Subgraphs of unit distance graph of $\mathcal{R}$ in Example 2 for Algorithm 2

The priority of separator $\mathcal{Z}^{(1,1)}$ is 2, and the priorities of the other separators are 1. Then the edges in $\mathcal{Z}^{(1,1)}$ are removed from the edge set $\mathcal{E}^{(1)}$. After these edges are removed, $\mathcal{Z}^{(1,3)}$ and $\mathcal{Z}^{(1,4)}$ are reconstructed and one conflicting edge remains in $G^{(2)}$ (see Figure 2.3(c)). The separators in this graph are $\mathcal{Z}^{(2,1)} = \mathcal{Z}^{(1,2)}$ and $\mathcal{Z}^{(2,2)} = \mathcal{Z}^{(1,5)}$. The priorities of both separators are 1, and either of them can be chosen for refinement. If $\mathcal{Z}^{(2,2)}$ is chosen, $\mathcal{Z}^{(2,1)}$ is reconstructed and no conflicting edges remain in $G^{(3)}$ (see Figure 2.3(d)). By Corollary 1, the vertices in each connected component of $G^{(3)}$ are the elements of the same partition of the Cartesian product partitioning. Therefore, the number of partitions in the Cartesian product partitioning of $\mathcal{R}$ is 4 and the partitions are given by $\{(0, 0, 1), (1, 0, 1), (2, 0, 1), (3, 0, 1)\}$, $\{(1, 0, 0), (2, 0, 0)\}$, $\{(3, 1, 0)\}$, and $\{(0, 1, 0), (0, 1, 1), (1, 1, 0), (1, 1, 1), (2, 1, 0), (2, 1, 1)\}$.

2.3 Experimental results

We implemented the proposed partitioning algorithms in C; the code can be obtained from [48]. All experiments are performed on a PC with an Intel Core i7 Quad-Core 3.6 GHz processor with 32 GB of main memory. All times are reported as seconds of CPU time. We monitor the memory allocation of the algorithms with the `pidstat` command of the `sysstat` package under Linux and all memory allocation results are reported as Megabytes (MB). We consider two groups of test problems. The first group consists of multi-dimensional reachable state spaces of Markovian models from the literature. The second group consists of a class of randomly generated multi-dimensional reachable state spaces with known minimum Cartesian product partitioning. We use the second group to find out how good the partitioning algorithms perform in terms of number of partitions with respect to the optimal solution.

2.3.1 Test problems from literature

We consider state spaces of call center models with different control policies (N-model, V-model, W-model) [30], a parallel communication software model (courier_large, courier_medium) [20], a manufacturing systems model with Kanban control (kanban_fail, kanban_large) [21], and a queueing model with multiple servers and queues (msmq_large) [26]. The V-model has different variations with 2, 3, and 4 types of customers that we consider. ‘V-model (2)’, ‘V-model (3)’ and ‘V-model (4)’ in the results refer to the V-model with 2, 3, and 4 customer types [30].

Table 2.1 gives the properties of the first group of test problems. The first four columns give the model name, the dimension of the state space, the size of the product state space, and the size of the reachable state space. The fifth column gives the minimum number of partitions in Cartesian product partitioning. The states of each connected component in the graph, that is obtained by removing the separators from the unit distance graph of $\mathcal{R}$, are in same partition in the

minimum Cartesian product partitioning of $\mathcal{R}$ (see Lemma 5 in [47]). Hence, each partition in the minimum Cartesian product partitioning of $\mathcal{R}$ is a union of partitions obtained by removing the separators from the unit distance graph of $\mathcal{R}$. We compute all possible Cartesian product partitionings satisfying this condition in order to obtain the number of partitions in the minimum Cartesian product partitioning. This is trivial when there are small number of separators and intersecting separator pairs. However, the number of Cartesian product partitionings to compute increases exponentially with the number of separators and intersecting separator pairs. We also report two important characteristics of the models. The sixth and seventh columns give the number of separators and the number of intersecting separator pairs, respectively. The reachability of a multi-dimensional state depends on the interaction of the subsystems. Hence, the number of separators and intersecting separator pairs depends on the model and the reachability conditions. There are many different reachability conditions, such as the sum of different subsystem state values might be bounded or a subsystem cannot be in a state depending on other subsystem states. In the N-model, the sum of different subsystem state values is bounded implying more separators and intersecting separator pairs. There is no such interaction in the other models we consider. V-model, W-model, courier_large, and courier_medium are simpler than the N-model, and have relatively small numbers of separators. There are no separators in kanban_fail, kanban_large, msmq_large, and kanban_large, that is, the unit distance graphs of these models do not include any conflicting edges. Therefore, these models can be considered as trivial.

Table 2.1: Properties of models from literature

Model	d	$ S $	$ \mathcal{R} $	$MinPrt$	Sep	$SepInt$
N-model	5	2,857,680	50,982	35	47	168
V-model (2)	3	2,695,784	103,709	2	3	3
V-model (3)	4	634,114	24,414	2	4	6
V-model (4)	5	260,000	10,025	2	5	10
W-model	5	1,377,810	20,142	4	5	10
courier_large	4	56,265,300	1,632,600	9	8	4
courier_medium	4	8,593,200	419,400	7	6	3
kanban_fail	4	19,131,876	2,302,911	8	0	0
kanban_large	4	1,742,400	1,742,400	1	0	0
msmq_large	7	1,280,000,000	1,311,744	28	0	0

In Table 2.2, we present the results of experiments for the models from literature when the states are processed in lexicographical order. If the states are not given in this order, then it may be easily obtained by sorting. The second and fifth columns give the number of partitions in the Cartesian product partitionings computed by merge and refinement based partitioning algorithms, respectively. When the states are processed in lexicographical order, both algorithms compute the optimal solution for the `courier_large`, `courier_medium`, `kanban_fail`, `kanban_large`, and `msmq_large` models. The merge based algorithm computes the optimal solution of only the N-model among the call center models. The refinement based algorithm computes the optimal solution for all call center models except the relatively complicated N-model. It is reasonable to use merge based algorithm for relatively complicated problems. For all the models, the merge based algorithm requires less time and memory than the refinement based algorithm.

Table 2.2: Experimental results for models from literature when states are processed in lexicographical order

Model	<i>Prt</i>	Merge		<i>Prt</i>	Refinement	
		Time	Memory		Time	Memory
N-model	35	0.2	0.9	43	0.1	21.4
V-model (2)	3	0.1	0.9	2	0.2	31.7
V-model (3)	4	0.0	0.9	2	0.1	10.4
V-model (4)	5	0.0	1.0	2	0.0	5.6
W-model	5	0.0	0.9	4	0.1	9.6
<code>courier_large</code>	9	2.1	0.9	9	4.8	576.4
<code>courier_medium</code>	7	0.5	0.9	7	1.2	149.4
<code>kanban_fail</code>	8	2.8	1.0	8	7.0	812.6
<code>kanban_large</code>	1	2.0	1.1	1	5.8	615.0
<code>msmq_large</code>	28	3.1	1.1	28	6.2	652.7

In Table 2.3, we present the results of experiments for the models from literature when the states are processed in random order. For both algorithms, we compute the Cartesian product partitioning by processing the states in 51 random orderings so as to provide mean values. The second and fifth columns of Table 2.3 give the mean number of partitions together with the confidence interval both rounded to the nearest integer for a confidence probability of 95% of the partitionings obtained by merge and refinement based algorithms, respectively. In ‘Time’ and ‘Memory’ columns, the mean values and the confidence intervals for

a confidence probability of 95% of the experiments are reported. In all columns, the first value in parentheses is the mean and the other value is the confidence interval.

Table 2.3: Experimental results for models from literature when states are processed in random order

Model	<i>Prt</i>	Merge		<i>Prt</i>	Refinement	
		Time	Memory		Time	Memory
N-model	(6,098; 46)	(0.2; 0.0)	(4.1; 0.0)	(43; 0)	(0.2; 0.0)	(21.4; 0.0)
V-model (2)	(12,500; 78)	(0.3; 0.0)	(6.5; 0.0)	(2; 0)	(0.3; 0.0)	(31.6; 0.0)
V-model (3)	(5,792; 33)	(0.1; 0.0)	(3.0; 0.0)	(2; 0)	(0.1; 0.0)	(10.4; 0.0)
V-model (4)	(2,825; 20)	(0.0; 0.0)	(2.2; 0.0)	(2; 0)	(0.0; 0.0)	(5.6; 0.0)
W-model	(4,715; 35)	(0.1; 0.0)	(2.7; 0.0)	(4; 0)	(0.1; 0.0)	(9.5; 0.0)
courier_large	(444,818; 254)	(13.6; 0.2)	(87.8; 0.1)	(9; 0)	(13.2; 0.2)	(576.7; 0.3)
courier_medium	(114,648; 124)	(2.5; 0.0)	(23.9; 0.1)	(7; 0)	(2.7; 0.0)	(149.4; 0.0)
kanban_fail	(626,586; 284)	(19.3; 0.2)	(123.0; 0.1)	(8; 0)	(18.2; 0.2)	(812.6; 0.0)
kanban_large	(483,245; 238)	(14.7; 0.2)	(94.6; 0.1)	(1; 0)	(13.8; 0.2)	(615.2; 0.3)
msmq_large	(432,018; 154)	(14.9; 0.2)	(98.4; 0.1)	(28; 0)	(16.1; 0.2)	(652.6; 0.0)

The partitioning size and the memory requirement remain the same for the refinement based algorithm, but the time requirement increases when the states are processed in random order. In all of the models, the number of separators is small, so the increase in time is not due to the refinement of the graph and updating of separators, but due to the graph construction. These results suggest that cache is more efficiently used while accessing the states when the states are ordered lexicographically since each state becomes a child of the last accessed state in the AVL tree. Hence, insertion to the tree becomes much more efficient for the lexicographic ordering of the states. The order of states is much more important for the merge based algorithm. In this algorithm, two partitions are merged along dimension h if the subsets of their subsystems are the same in all dimensions except h . When the states are ordered lexicographically, a partition is merged with all possible partitions along a dimension before it is merged with a partition in another dimension. When the states are ordered randomly, partitions are merged along random dimensions depending on the order of the states. Hence, all possible partitions along a dimension are not considered. Therefore, the number of partitions for random ordering of the states increases by a factor between 174 (N-model) and 483,000 (kanban_large) on average with respect to

the number of partitions for the lexicographical ordering. The time and memory requirements also increase substantially which agrees with the complexity analysis since the time and memory complexities of the merge based algorithm depend on the number of partitions (see Section 2.2.1). The time taken by the merge based algorithm becomes close to the time taken by the refinement based algorithm. However, the memory requirement of the merge based algorithm is still better than the memory requirement of the refinement based algorithm.

2.3.2 A class of random test problems

Cartesian product partitioning of a 2-dimensional reachable state space is equivalent to the hyper-rectangular partitioning of the 2-dimensional polytope that is represented by the reachable state space. Hence, minimum Cartesian product partitioning of a 2-dimensional reachable state space can be obtained by using a modified version of the minimum rectangular partitioning algorithm in [69]. Consider the next lemma that specifies a condition for increasing the dimension of the state space in Cartesian product partitioning without changing the minimum number of partitions.

Lemma 4. Let $L \in \mathbb{Z}_{>0}$ and let $\{\mathcal{R}^{(1)}, \dots, \mathcal{R}^{(L)}\}$ be the minimum Cartesian product partitioning of $\cup_{l=1}^L \mathcal{R}^{(l)} \subseteq \mathbb{Z}_{\geq 0}^d$. Then the minimum Cartesian product partitioning of $\cup_{l=1}^L (\mathcal{R}^{(l)} \times \mathcal{Q}^{(l)}) \subseteq \mathbb{Z}_{\geq 0}^{d+1}$ is $\{\mathcal{R}^{(1)} \times \mathcal{Q}^{(1)}, \dots, \mathcal{R}^{(L)} \times \mathcal{Q}^{(L)}\}$ if $\cap_{l=1}^L \mathcal{Q}^{(l)} \neq \emptyset$ and $\mathcal{Q}^{(l)} \subseteq \mathbb{Z}_{\geq 0}$ consists of consecutive integers for $l = 1, \dots, L$.

Proof. The sets $\mathcal{R}^{(l)} \times \mathcal{Q}^{(l)}$ and $\mathcal{R}^{(l')} \times \mathcal{Q}^{(l')}$ are disjoint since $\mathcal{R}^{(l)}$ and $\mathcal{R}^{(l')}$ are disjoint sets for $l \neq l'$ and $l, l' = 1, \dots, L$. Each set $\mathcal{R}^{(l)} \times \mathcal{Q}^{(l)}$ is a Cartesian product of sets of consecutive integers for $l = 1, \dots, L$. Therefore,

$$\{\mathcal{R}^{(1)} \times \mathcal{Q}^{(1)}, \dots, \mathcal{R}^{(L)} \times \mathcal{Q}^{(L)}\}$$

is a Cartesian product partitioning of $\cup_{l=1}^L (\mathcal{R}^{(l)} \times \mathcal{Q}^{(l)})$. Now, we show that the size of the minimum Cartesian product partitioning must be L .

Assume that $\{\mathcal{P}^{(1)}, \dots, \mathcal{P}^{(M)}\}$ is a Cartesian product partitioning of

$\cup_{l=1}^L(\mathcal{R}^{(l)} \times \mathcal{Q}^{(l)})$, where $M < L$ and $\mathcal{P}^{(m)} = \times_{h=1}^{d+1} \mathcal{P}_h^{(m)}$ for $m = 1, \dots, M$. Since $\cap_{l=1}^L \mathcal{Q}^{(l)} \neq \emptyset$, there exists some $q \in \cap_{l=1}^L \mathcal{Q}^{(l)}$. Then $\cup_{l=1}^L(\mathcal{R}^{(l)} \times \{q\})$ can be partitioned into M or less Cartesian products. In this case, $\cup_{l=1}^L \mathcal{R}^{(l)}$ can also be partitioned into M or less partitions. However, this contradicts the assumption that $\{\mathcal{R}^{(1)}, \dots, \mathcal{R}^{(L)}\}$ is the minimum Cartesian product partitioning of $\cup_{l=1}^L \mathcal{R}^{(l)}$. $\square$

Lemma 4 states that when the minimum Cartesian product partitioning of a multi-dimensional state space is known, it is possible to add states and increase the dimension without changing the size of the minimum Cartesian product partitioning.

Using Lemma 4, Algorithm 3 presented next generates a random multi-dimensional state space whose minimum Cartesian product partitioning is known. The algorithm starts with a random 2-dimensional state space over $\{0, \dots, M-1\} \times \{0, \dots, M-1\}$ that is generated by assigning each state a reachability probability p (lines 3-4). Then the minimum Cartesian product partitioning of this random set is computed (lines 6-7). Next, $(d-2)$ dimensions are added to the state space one at a time (lines 8-14). For each additional dimension, a uniformly distributed random integer that determines the range of consecutive integers is generated (line 9). Then the Cartesian product of each partition with the set of consecutive integers is computed and the corresponding integer is updated to continue with the next dimension (lines 11-12). Finally, dimensions of the states in $\mathcal{R}$ are rearranged so that the initial two dimensions are not always the first two dimensions (line 18).

Table 2.4 reports the properties of the randomly generated state spaces. The first three columns provide the parameters used in generating the test problems. The first two columns give the inputs of Algorithm 3, p and d , respectively. The third column gives the size of the product state space, L^d , when each subsystem state space size is L . The remaining columns have the same meaning as in Table 2.4, where mean values of 51 random test problems are reported for each set of parameters. Observe that the properties of the reachable state space depend on

Input: Dimension of reachable state space: d
State space size of each subsystem: M
Probability of a state to be reachable in initial 2-dimensional state space: p

Output: d -dimensional reachable state space: $\mathcal{R}$
Size of minimum Cartesian product partitioning of $\mathcal{R}$: L

```

1: function GENERATERANDOMTESTPROBLEM( $d, M, p, \mathcal{R}, L$ )
2:    $\mathcal{R}' \leftarrow \emptyset$ 
3:   for all  $(i, j) \in \{0, \dots, M-1\} \times \{0, \dots, M-1\}$  do
4:      $\mathcal{R}' \leftarrow \mathcal{R}' \cup \{(i, j)\}$  with probability  $p$ 
5:   end for
6:   Compute minimum Cartesian product partitioning of  $\mathcal{R}'$ ,  $\{\mathcal{R}^{(1)}, \dots, \mathcal{R}^{(L)}\}$  by
7:     using a modified version of minimum rectangular partitioning algorithm [69]
8:   for all  $h = 3, \dots, d$  do
9:      $m \leftarrow \text{Unif}(0, M-1)$ 
10:    for all  $l = 1, \dots, L$  do
11:       $a \leftarrow \text{Unif}(0, m)$ ;  $b \leftarrow \text{Unif}(m, M-1)$ 
12:       $\mathcal{R}^{(l)} \leftarrow \mathcal{R}^{(l)} \times \{a, \dots, b\}$ 
13:    end for
14:  end for
15:  for all  $l = 1, \dots, L$  do
16:     $\mathcal{R} \leftarrow \mathcal{R} \cup \mathcal{R}^{(l)}$ 
17:  end for
18:  Rearrange the dimensions of the states in  $\mathcal{R}$  with a new random ordering
19: end function

```

Algorithm 3: Algorithm that generates a random multi-dimensional state space with known minimum Cartesian product partitioning

the initial 2-dimensional state space. As p and L increase, the number of separators and the number of intersecting pairs increase. As these numbers increase, the problems become more complicated and it tends to be more difficult to find separators leading to a better solution. Observe that the size of the partitioning, L , depends on $\mathcal{R}'$ (line 6 of Algorithm 3). When p is small, there are less reachable states to be merged. When p is large, there are less conflicting edges in the unit distance graph of $\mathcal{R}'$. Therefore, the size of the partitioning does not increase as p increases.

In Table 2.5, we present the results of experiments with the randomly generated state spaces of Table 2.4. The first three columns have the same meaning as in Table 2.4 and the remaining columns have the same meaning as in Table 2.3. When p is 0.25 and 0.50, the sizes of the partitionings obtained by the refinement based algorithm are close to their minimum values. However, when p is 0.75, the number of partitions become larger as the size of the product state space increases

Table 2.4: Properties of random test problems

p	d	$ \mathcal{S} $	$ \mathcal{R} $	$MinPrt$	Sep	$SepInt$
0.25	3	10^3	(134; 14)	(17; 1)	(10; 3)	(11; 4)
		20^3	(1,059; 62)	(66; 3)	(46; 8)	(60; 14)
		30^3	(3,525; 123)	(148; 4)	(110; 12)	(149; 27)
	4	10^4	(719; 109)	(17; 2)	(15; 5)	(27; 13)
		20^4	(10,719; 809)	(66; 3)	(64; 12)	(129; 33)
		30^4	(54,075; 2,069)	(148; 4)	(156; 19)	(327; 60)
	5	10^5	(4,247; 682)	(16; 1)	(18; 5)	(38; 15)
		20^5	(117,901; 10,960)	(67; 3)	(85; 16)	(212; 56)
		30^5	(856,358; 49,238)	(148; 5)	(204; 23)	(589; 128)
	3	10^3	(272; 20)	(23; 1)	(44; 5)	(88; 22)
		20^3	(2,090; 100)	(87; 2)	(204; 16)	(566; 97)
		30^3	(7,016; 212)	(195; 3)	(484; 25)	(1,517; 188)
	4	10^4	(1,484; 169)	(23; 1)	(61; 8)	(197; 49)
		20^4	(22,146; 1,379)	(88; 3)	(284; 18)	(1,325; 207)
		30^4	(108,987; 3,902)	(196; 4)	(677; 35)	(4,046; 540)
0.50	5	10^5	(8,629; 1,341)	(23; 1)	(75; 8)	(351; 74)
		20^5	(232,892; 15,774)	(88; 3)	(352; 25)	(2,572; 482)
		30^5	(1,626,489; 82,115)	(196; 4)	(870; 47)	(7,664; 921)
	3	10^3	(392; 32)	(21; 1)	(65; 4)	(227; 34)
		20^3	(3,249; 144)	(72; 3)	(261; 9)	(1,720; 90)
		30^3	(10,587; 208)	(161; 3)	(590; 17)	(5,418; 270)
	4	10^4	(2,285; 264)	(21; 1)	(83; 6)	(518; 56)
		20^4	(33,827; 2,004)	(75; 3)	(339; 15)	(4,201; 286)
		30^4	(162,781; 7,263)	(162; 4)	(765; 21)	(13,607; 628)
	5	10^5	(12,101; 1,693)	(20; 1)	(99; 7)	(863; 93)
		20^5	(354,698; 26,549)	(75; 2)	(412; 17)	(7,590; 480)
		30^5	(2,560,044; 163,475)	(160; 3)	(915; 28)	(24,969; 1,364)

when number of separators and intersecting separator pairs increase. As for the refinement based algorithm, the number of partitions obtained by the merge based algorithm is closer to the optimal value for smaller p . For all test problems, the merge based algorithm is better than the refinement based algorithm in terms of time and memory requirements.

Table 2.5: Experimental results for random test problems

p	d	$ \mathcal{S} $	Prt	Merge		Prt	Refinement	
				Time	Memory		Time	Memory
0.25	3	10^3	(17; 2)	(0.0; 0.0)	(0.0; 0.0)	(17; 1)	(0.0; 0.0)	(0.0; 0.0)
		20^3	(70; 4)	(0.0; 0.0)	(0.6; 0.2)	(67; 3)	(0.0; 0.0)	(1.5; 0.4)
		30^3	(154; 5)	(0.0; 0.0)	(0.9; 0.0)	(148; 4)	(0.0; 0.0)	(2.7; 0.1)
	4	10^4	(17; 2)	(0.0; 0.0)	(0.5; 0.3)	(17; 2)	(0.0; 0.0)	(1.4; 0.5)
		20^4	(69; 5)	(0.0; 0.0)	(0.9; 0.0)	(66; 3)	(0.0; 0.0)	(5.6; 0.3)
		30^4	(156; 8)	(0.1; 0.0)	(0.9; 0.0)	(148; 4)	(0.1; 0.0)	(20.8; 0.7)
	5	10^5	(18; 2)	(0.0; 0.0)	(0.9; 0.0)	(16; 1)	(0.0; 0.0)	(3.3; 0.4)
		20^5	(73; 6)	(0.2; 0.0)	(1.0; 0.0)	(67; 3)	(0.4; 0.0)	(47.1; 4.2)
		30^5	(155; 11)	(1.9; 0.1)	(0.9; 0.0)	(148; 5)	(3.9; 0.3)	(330.7; 18.9)
0.50	3	10^3	(26; 2)	(0.0; 0.0)	(0.0; 0.0)	(23; 1)	(0.0; 0.0)	(0.0; 0.0)
		20^3	(101; 7)	(0.0; 0.0)	(0.8; 0.2)	(87; 2)	(0.0; 0.0)	(1.9; 0.5)
		30^3	(224; 15)	(0.0; 0.0)	(0.9; 0.0)	(197; 4)	(0.0; 0.0)	(3.8; 0.1)
	4	10^4	(28; 4)	(0.0; 0.0)	(0.8; 0.2)	(23; 1)	(0.0; 0.0)	(1.9; 0.5)
		20^4	(106; 14)	(0.0; 0.0)	(0.9; 0.0)	(89; 3)	(0.1; 0.0)	(9.6; 0.5)
		30^4	(241; 25)	(0.2; 0.0)	(0.9; 0.0)	(197; 4)	(0.4; 0.0)	(40.2; 1.4)
	5	10^5	(27; 4)	(0.0; 0.0)	(0.9; 0.0)	(23; 1)	(0.0; 0.0)	(5.1; 0.5)
		20^5	(117; 20)	(0.5; 0.0)	(0.9; 0.0)	(89; 3)	(1.2; 0.1)	(91.3; 6.0)
		30^5	(254; 47)	(3.8; 0.2)	(1.0; 0.2)	(198; 4)	(10.3; 0.7)	(626.8; 31.5)
0.75	3	10^3	(29; 4)	(0.0; 0.0)	(0.0; 0.0)	(23; 3)	(0.0; 0.0)	(0.4; 0.4)
		20^3	(106; 13)	(0.0; 0.0)	(0.9; 0.1)	(122; 36)	(0.0; 0.0)	(2.6; 0.2)
		30^3	(228; 31)	(0.0; 0.0)	(0.9; 0.0)	(619; 183)	(0.0; 0.0)	(4.8; 0.1)
	4	10^4	(35; 7)	(0.0; 0.0)	(0.8; 0.2)	(21; 2)	(0.0; 0.0)	(2.5; 0.1)
		20^4	(129; 26)	(0.1; 0.0)	(0.9; 0.0)	(94; 28)	(0.2; 0.0)	(13.7; 0.7)
		30^4	(240; 65)	(0.3; 0.0)	(1.0; 0.1)	(421; 240)	(2.0; 0.2)	(59.2; 2.6)
	5	10^5	(37; 9)	(0.0; 0.0)	(1.0; 0.0)	(21; 2)	(0.0; 0.0)	(6.5; 0.7)
		20^5	(113; 41)	(0.7; 0.1)	(0.9; 0.0)	(82; 18)	(6.0; 0.9)	(138.5; 10.2)
		30^5	(307; 104)	(6.0; 0.5)	(1.1; 0.2)	(287; 112)	(92.0; 13.4)	(987.9; 62.9)

Chapter 3

On vector–Kronecker product multiplication

The generator matrix needs to be kept in compact form during analysis; hence, numerical techniques require multiplying vectors with Kronecker product efficiently. To this end, an efficient vector–Kronecker product multiplication algorithm based on shuffle algebra [42] and known as the shuffle algorithm [129] has been devised and used. In this algorithm, the multiplication of a vector with the Kronecker product is achieved by multiplying the vector with as many matrices as the number of factors, say d , in the Kronecker product, where each of the d matrices is expressed as the Kronecker product of $(d - 1)$ identity matrices and a different factor whose order among the identity matrices is that of the same factor in the original Kronecker product, hence, the term shuffle. Although this approach does not alleviate the problem of having to store large solution vectors during analysis, it is still more memory efficient compared to conventional techniques that use sparse vector–matrix multiplication. Regarding time complexity, the shuffle algorithm is slower unless the factors in the Kronecker product become relatively dense.

Vector–Kronecker product multiplication algorithms that are of a different nature have been proposed in [24]. In the algorithms therein, nonzero elements

of the generator matrix are obtained on-the-fly and multiplied with the corresponding elements of the vector. These algorithms are devised for square factors, but they can be extended to handle rectangular factors without much difficulty. Pot-RwCl is the most efficient among the algorithms in [24] since it fully exploits common nonzero factors forming the nonzero elements of the sparse matrix corresponding to each Kronecker product term. Due to this, Pot-RwCl becomes more efficient than the shuffle algorithm when the factors in the Kronecker product terms are relatively sparse.

In this chapter, we aim at improving the efficiency of the shuffle algorithm for rectangular factors. We implement the shuffle algorithm in such a way that zero rows and columns in factors of Kronecker product terms are omitted during multiplication. This enables us to avoid flops that evaluate to zero during the course of the multiplication and to possibly reduce the amount of memory used. The shuffle algorithm, the Pot-RwCl algorithm, and the modified shuffle algorithm are compared analytically for their flop and memory requirements. Although the modification on the shuffle algorithm may seem simple, it is not intuitive (since zero rows and columns are not stored in sparse matrices anyway), but as we shall see through numerical experiments, it turns out to be quite effective in many cases. At the end, we are able to identify those implementations of vector-Kronecker product multiplication that should be preferred over others.

Note that we treat the Kronecker product factors as purely algebraic quantities and do not use their stochastic properties. Therefore, the proposed modification on the shuffle algorithm is not limited to the context of MCs. Moreover, the idea of avoiding some flops by omitting zero rows and columns in factors of Kronecker product terms is not limited to the improvement on the shuffle algorithm. A potential use of the proposed approach is in the semi-tensor product operation that is introduced recently to generalize matrix multiplication [37]. The semi-tensor product of two matrices is defined as the multiplication of Kronecker products of identity matrices and these factors. It should be possible to devise an algorithm based on shuffle algebra to multiply a vector with a semi-tensor product of matrices without computing the semi-tensor product of the matrices. When this is the case, unnecessary flops can be avoided by omitting zero rows and columns in

the factors during the course of the multiplication.

The next two sections provide the shuffle and Pot-RwCl algorithms for rectangular factors and elaborates on how they work using a small example. The third section introduces the proposed modification on the shuffle algorithm. The merit of this approach is shown on the same small example with the help of a lemma. The fourth section discusses implementation issues associated with shuffle, Pot-RwCl, and modified shuffle algorithms. The fifth section presents numerical results on a number of benchmark models and comments on them.

3.1 Shuffle algorithm

In numerical methods used for analyzing Kronecker based MCs, vectors are multiplied with Kronecker product terms. Now, let $\mathbf{X}_h \in \mathbb{R}^{m_h \times n_h}$ for $h = 1, \dots, d$ be the rectangular factors in a given Kronecker product term. Then the shuffle algorithm is based on the identity

$$\bigotimes_{h=1}^d \mathbf{X}_h = \prod_{h=1}^d (\mathbf{I}_{\prod_{f=1}^{h-1} n_f} \otimes \mathbf{X}_h \otimes \mathbf{I}_{\prod_{f=h+1}^d m_f}),$$

which is due to compatibility of Kronecker product with matrix multiplication [68]. Besides, $\mathbf{I}_{\prod_{f=1}^{h-1} n_f} \otimes \mathbf{X}_h \otimes \mathbf{I}_{\prod_{f=h+1}^d m_f}$ is an identity matrix if $\mathbf{X}_h = \mathbf{I}_{m_h}$. Hence, the left-multiplication of $\mathbf{p} \in \mathbb{R}^{1 \times \prod_{h=1}^d m_h}$ with $\mathbf{X} = \bigotimes_{h=1}^d \mathbf{X}_h$ can be accomplished as in Algorithm 4. Note that this results in an output vector whose length ranges from $n_1 \prod_{h=2}^d m_h$ to $\prod_{h=1}^d n_h$ during the course of the multiplication.

Now, observe that the only flops executed in Algorithm 4 are in line 13, where the temporary vector $\mathbf{z}$ is multiplied with the nonidentity matrix $\mathbf{X}_h$. For a fixed value of h , this line is executed $\prod_{f=1}^{h-1} n_f \prod_{f=h+1}^d m_f$ times, and the cost of vector multiplication with $\bigotimes_{h=1}^d \mathbf{X}_h$ using the shuffle algorithm amounts to

$$2 \sum_{h \in \mathcal{H}} nnz(\mathbf{X}_h) \prod_{f=1}^{h-1} n_f \prod_{f=h+1}^d m_f \quad (3.1)$$

flops, where

$$\mathcal{H} = \{h \in \{1, \dots, d\} \mid \mathbf{X}_h \neq \mathbf{I}_{m_h}\}.$$

```

Input: Vector:  $\mathbf{p}$ 
          Rectangular Kronecker product factors:  $\mathbf{X}_1, \dots, \mathbf{X}_d$ 
Output: Vector:  $\mathbf{q} = \mathbf{p} \otimes_{h=1}^d \mathbf{X}_h$ :
1: function SHUFFLE( $\mathbf{p}, \mathbf{X}_1, \dots, \mathbf{X}_d, \mathbf{q}$ )
2:   Copy  $\mathbf{p}$  to  $\mathbf{q}$ ;
3:    $i_{left} = 1; i_{right} = \prod_{h=2}^d m_h; m_{d+1} = 1;$ 
4:   for all  $h = 1$  to  $d$  do
5:     if  $\mathbf{X}_h \neq \mathbf{I}_{m_h}$  then
6:        $base_i = 0; base_j = 0;$ 
7:       for all  $i_l = 0, \dots, i_{left} - 1$  do
8:         for all  $i_r = 0, \dots, i_{right} - 1$  do
9:            $index_i = base_i + i_r;$ 
10:          for all  $row = 0, \dots, m_h - 1$  do
11:             $\mathbf{z}(row) = \mathbf{q}(index_i); index_i = index_i + i_{right};$ 
12:          end for
13:           $\mathbf{z}' = \mathbf{z}\mathbf{X}_h;$ 
14:           $index_j = base_j + i_r;$ 
15:          for all  $col = 0, \dots, n_h - 1$  do
16:             $\mathbf{q}'(index_j) = \mathbf{z}'(col); index_j = index_j + i_{right};$ 
17:          end for
18:        end for
19:         $base_i = base_i + m_h i_{right}; base_j = base_j + n_h i_{right};$ 
20:      end for
21:      Copy  $\mathbf{q}'$  to  $\mathbf{q}$ 
22:    end if
23:     $i_{left} = i_{left} n_h; i_{right} = i_{right} / m_{h+1};$ 
24:  end for
25: end function

```

Algorithm 4: Shuffle algorithm for rectangular factors

When $|\mathcal{H}| = 1$, we have that $m_h = n_h = nnz(\mathbf{X}_h)$ for $h \notin \mathcal{H}$ and $h = 1, \dots, d$, and therefore, the number of flops executed by the shuffle algorithm is $2nnz(\mathbf{X})$

Algorithm 4 requires four vectors: $\mathbf{q}$, $\mathbf{q}'$, $\mathbf{z}$, and $\mathbf{z}'$ in addition to the input vector. The vectors $\mathbf{q}$ and $\mathbf{q}'$ are the floating-point vectors storing the intermediate results; hence, each of them needs to be as long as $\max_{h \in (\{0\} \cup \mathcal{H})} (\prod_{f=1}^h n_f \prod_{f=h+1}^d m_f)$. When the input vector needs to be multiplied with sums of Kronecker product terms, the output vector cannot be used to keep intermediate results. The floating-point vectors $\mathbf{z}$ and $\mathbf{z}'$ need to be at least as long as $\max_h(m_h)$ and $\max_h(n_h)$, respectively.

The next example is given to show how vector-Kronecker product multiplication works in the presence of rectangular factors when Algorithm 4 is used.

Example 3. Consider the multiplication of the vector

$$\mathbf{p} = [a_0 \ a_1 \ a_2 \ a_3 \ a_4 \ a_5 \ a_6 \ a_7 \ a_8 \ a_9 \ a_{10} \ a_{11} \ a_{12} \ a_{13} \ a_{14} \ a_{15} \ a_{16} \ a_{17}]$$

with the Kronecker product term $\mathbf{X} = \mathbf{X}_1 \otimes \mathbf{X}_2 \otimes \mathbf{X}_3$, where

$$\mathbf{X}_1 = \begin{bmatrix} 3 \\ 2 \end{bmatrix}_{3 \times 2}, \quad \mathbf{X}_2 = \begin{bmatrix} 2 \end{bmatrix}_{3 \times 2}, \quad \text{and} \quad \mathbf{X}_3 = \begin{bmatrix} 5 \\ 1 \ 3 \end{bmatrix}_{2 \times 3}.$$

In Algorithm 4, the outer loop in line 4 is executed three times for $h = 1, 2, 3$. At the beginning of the algorithm, $\mathbf{p}$ is copied to $\mathbf{q}$ and $\mathbf{q}' = \mathbf{q}(\mathbf{X}_1 \otimes \mathbf{I}_6)$ is computed for $h = 1$. In this turn, $nnz(\mathbf{X}_1) = 2$, $i_{left} = 1$, and $i_{right} = 6$. Therefore, the middle loop in line 7 and the inner loop in line 8 are executed once and six times, respectively. Then $\mathbf{q}(\mathbf{X}_1 \otimes \mathbf{I}_6)$ is computed as

$$\mathbf{q}' = [3a_0 + 2a_6 \ 3a_1 + 2a_7 \ 3a_2 + 2a_8 \ 3a_3 + 2a_9 \ 3a_4 + 2a_{10} \ 3a_5 + 2a_{11} \ 0 \ 0 \ 0 \ 0 \ 0 \ 0]$$

in 24 flops and $\mathbf{q}'$ is copied to $\mathbf{q}$ at the end of the first turn. In the second turn, $\mathbf{q}' = \mathbf{q}(\mathbf{I}_2 \otimes \mathbf{X}_2 \otimes \mathbf{I}_2)$ is computed for $h = 2$. In this turn, $nnz(\mathbf{X}_2) = 1$, $i_{left} = 2$, and $i_{right} = 2$. Therefore, the middle loop and the inner loop are each executed twice. Then $\mathbf{q}(\mathbf{I}_2 \otimes \mathbf{X}_2 \otimes \mathbf{I}_2)$ is computed as

$$\mathbf{q}' = [0 \ 0 \ 6a_2 + 4a_8 \ 6a_3 + 4a_9 \ 0 \ 0 \ 0 \ 0]$$

in 8 flops and $\mathbf{q}'$ is copied to $\mathbf{q}$ at the end of the second turn. In the last turn, $\mathbf{q}' = \mathbf{q}(\mathbf{I}_4 \otimes \mathbf{X}_3)$ is computed for $h = 3$. In this turn, $nnz(\mathbf{X}_3) = 3$, $i_{left} = 4$, and $i_{right} = 1$. Therefore, the middle loop and the inner loop are executed four times and once, respectively. Then $\mathbf{q}(\mathbf{I}_4 \otimes \mathbf{X}_3)$ is computed as

$$\mathbf{q}' = [0 \ 0 \ 0 \ 0 \ 30a_2 + 6a_3 + 20a_8 + 4a_9 \ 18a_3 + 12a_9 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0]$$

in 24 flops, and $\mathbf{q}'$ is copied back to $\mathbf{q}$ at the end of the last turn. Hence, Algorithm 4's computation of $\mathbf{q} = \mathbf{p}\mathbf{X}$ takes altogether 56 flops.

3.2 Pot-RwCl algorithm

A second approach for computing $\mathbf{p} \otimes_{h=1}^d \mathbf{X}_h$ is to generate nonzero elements of the Kronecker product term and multiply them with corresponding elements

of the vector. A nonzero element of the Kronecker product term is obtained as a result of the multiplication of d nonzeros each coming from a different factor. The Pot-RwCl algorithm proposed in [24] exploits common multipliers of nonzero elements of Kronecker product terms so that multiplications with the same values are avoided. Hence, $\mathbf{p} \otimes_{h=1}^d \mathbf{X}_h$ can be computed recursively by calling Algorithm 5 with the initial parameters $h = 1$, $i = 0$, $j = 0$, and $v = 1$.

Input: Vector: $\mathbf{p}$
 Rectangular Kronecker product factors: $\mathbf{X}_1, \dots, \mathbf{X}_d$
 Subsystem: h
 Row index obtained from submatrices $1, \dots, h-1$: i
 Column index obtained from submatrices $1, \dots, h-1$: j
 Nonzero obtained from submatrices $1, \dots, h-1$: v
Output: Vector: $\mathbf{q}$ such that $\mathbf{q}(\mathcal{J}') = \mathbf{q}(\mathcal{J}) + \mathbf{p}(\mathcal{I}') (v \otimes_{f=h}^d \mathbf{X}_f)$, where
 $\mathcal{I}' = \{i, \dots, i + \prod_{f=h}^d m_f - 1\}$ and $\mathcal{J}' = \{j, \dots, j + \prod_{f=h}^d n_f - 1\}$
 1: **function** POT-RWCL($\mathbf{p}, \mathbf{X}_1, \dots, \mathbf{X}_d, h, i, j, v, \mathbf{q}$)
 2: **for all** (i_h, j_h) such that $\mathbf{X}_h(i_h, j_h) > 0$ **do**
 3: $i' \leftarrow i + i_h \prod_{f=h+1}^d m_f$; $j' \leftarrow j + j_h \prod_{f=h+1}^d n_f$;
 4: **if** $\mathbf{X}_h \neq \mathbf{I}_{m_h}$ **then**
 5: $v' \leftarrow v \mathbf{X}_h(i_h, j_h)$;
 6: **end if**
 7: **if** $h < d$ **then**
 8: POT-RWCL($\mathbf{p}, \mathbf{X}_1, \dots, \mathbf{X}_d, h+1, i', j', v', \mathbf{q}$)
 9: **else**
 10: $\mathbf{q}(j') = \mathbf{q}(j') + \mathbf{p}(i') v'$;
 11: **end if**
 12: **end for**
 13: **end function**

Algorithm 5: Pot-RwCl algorithm for rectangular factors

Observe that the only flops executed in Algorithm 5 are in lines 5 and 10. In line 5, multiplication of nonzeros from subsystems 1 through h are performed. For a fixed value of h , the function is called $\prod_{f=1}^{h-1} nnz(\mathbf{X}_f)$ times. Hence, the number of flops executed in line 5 is $\prod_{f=1}^h nnz(\mathbf{X}_f)$ if $\mathbf{X}_h \neq \mathbf{I}_{m_h}$. On the other hand, line 10 is executed for each nonzero element of the Kronecker product of factors. Therefore, the cost of vector-Kronecker product multiplication using the Pot-RwCl algorithm amounts to

$$\sum_{h \in \mathcal{H}} \prod_{f=1}^h nnz(\mathbf{X}_f) + 2 nnz(\mathbf{X}) \quad (3.2)$$

flops. Since Algorithm 5 does not have any intermediate vector computations, it does not require any additional floating-point vectors.

Next, we demonstrate how Pot-RwCl works using our running example.

Example 3. (cont'd) The matrix $\mathbf{X} = \mathbf{X}_1 \otimes \mathbf{X}_2 \otimes \mathbf{X}_3$ includes 6 nonzero elements which can be written as

$$\mathbf{X}(2, 4) = \mathbf{X}_1(0, 0)\mathbf{X}_2(1, 1)\mathbf{X}_3(0, 1), \quad \mathbf{X}(3, 4) = \mathbf{X}_1(0, 0)\mathbf{X}_2(1, 1)\mathbf{X}_3(1, 1),$$

$$\mathbf{X}(3, 5) = \mathbf{X}_1(0, 0)\mathbf{X}_2(1, 1)\mathbf{X}_3(1, 2), \quad \mathbf{X}(8, 4) = \mathbf{X}_1(1, 0)\mathbf{X}_2(1, 1)\mathbf{X}_3(0, 1),$$

$$\mathbf{X}(9, 4) = \mathbf{X}_1(1, 0)\mathbf{X}_2(1, 1)\mathbf{X}_3(1, 1), \quad \mathbf{X}(9, 5) = \mathbf{X}_1(1, 0)\mathbf{X}_2(1, 1)\mathbf{X}_3(1, 2),$$

where $\mathbf{X}_1(0, 0) = 3$, $\mathbf{X}_1(1, 0) = 2$, $\mathbf{X}_2(1, 1) = 2$, $\mathbf{X}_3(0, 1) = 5$, $\mathbf{X}_3(1, 1) = 1$, and $\mathbf{X}_3(1, 2) = 3$. Initially the function is called for the values $h = 1$, $i = 0$, $j = 0$, and $v = 1$. Since $\mathbf{X}_1$ includes two nonzeros, the loop in line 2 is executed twice. In the first turn, the function in line 8 is called for the values $h = 2$, $i' = 0$, $j' = 0$, and $v' = 3$. Then the loop in line 2 is executed once for the single nonzero element of $\mathbf{X}_2$ and the function in line 8 is called for the values $h = 3$, $i' = 2$, $j' = 3$, and $v' = 6$. Since $\mathbf{X}_3$ includes three nonzero elements, the loop in line 2 is executed three times. In these turns, $\mathbf{X}(2, 4) = 30$, $\mathbf{X}(3, 4) = 6$, and $\mathbf{X}(3, 5) = 18$ are obtained and $\mathbf{q}$ is computed as

$$\mathbf{q} = [0 \ 0 \ 0 \ 0 \ 30a_2 + 6a_3 \ 18a_3 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0]$$

in 11 flops. The second turn of the loop in line 2 of the function is executed for $\mathbf{X}_1(1, 0)$ and the function in line 8 is called for the values $h = 2$, $i' = 6$, $j' = 0$, and $v' = 2$. Then the loop in line 2 is executed once for the single nonzero element of $\mathbf{X}_2$ and the function in line 8 is called for the values $h = 3$, $i' = 8$, $j' = 3$, and $v' = 4$. Then $\mathbf{X}(8, 4) = 20$, $\mathbf{X}(9, 4) = 4$, and $\mathbf{X}(9, 5) = 12$ are obtained and $\mathbf{q}$ is computed as

$$\mathbf{q} = [0 \ 0 \ 0 \ 0 \ 30a_2 + 6a_3 + 20a_8 + 4a_9 \ 18a_3 + 12a_9 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0]$$

in 22 flops.

The next section introduces the proposed modification on the shuffle algorithm and analyzes its merit through a lemma.

3.3 Modified shuffle algorithm

In the shuffle algorithm, the number of flops executed due to a given factor does not depend on the nonzero structure of the other factors in the Kronecker product. However, when other factors include zero rows or columns, some vector elements end up being computed even though they would evaluate to zero at the end. A speculative example is the multiplication of a vector with a Kronecker product term including a zero factor. The shuffle algorithm executes possibly a large number of flops even though the result vector evaluates to zero at the end.

In this section, we present the proposed modification to the shuffle algorithm so that only relevant elements of the vector are multiplied with the nonzeros of factors and unnecessary flops are avoided. The following lemma provides the identity on which the modification is based.

Lemma 5. Let $\mathbf{P}_{(u, \mathcal{U})} \in \{0, 1\}^{u \times |\mathcal{U}|}$ be the matrix given elementwise as

$$\mathbf{P}_{(u, \mathcal{U})}(i, j) \begin{cases} 1 & \text{if } j = f^{(\mathcal{U})}(i) \text{ and } i \in \mathcal{U} \\ 0 & \text{otherwise} \end{cases},$$

where u is a finite positive integer, $\mathcal{U} \subseteq \{0, \dots, u-1\}$ is a nonempty set, and

$$f^{(\mathcal{U})}(i) = |\{j \in \mathcal{U} \mid j < i\}| \text{ for } i \in \mathcal{U}.$$

Besides, let $\mathbf{X}_h \in \mathbb{R}^{m_h \times n_h}$, $\mathcal{M}_h \subseteq \{0, \dots, m_h-1\}$ and $\mathcal{N}_h \subseteq \{0, \dots, n_h-1\}$ denote the sets of rows and columns that include at least one nonzero in $\mathbf{X}_h$ for $h = 1, \dots, d$. Then

$$\bigotimes_{h=1}^d \mathbf{X}_h = \bigotimes_{h=1}^d \mathbf{P}_{(m_h, \mathcal{M}_h)} \bigotimes_{h=1}^d \hat{\mathbf{X}}_h \bigotimes_{h=1}^d \mathbf{P}_{(n_h, \mathcal{N}_h)}^T, \quad (3.3)$$

where

$$\hat{\mathbf{X}}_h = \mathbf{P}_{(m_h, \mathcal{M}_h)}^T \mathbf{X}_h \mathbf{P}_{(n_h, \mathcal{N}_h)} \text{ for } h = 1, \dots, d.$$

Proof. Observe that $\bar{\mathbf{P}}_{(u, \mathcal{U})} = \mathbf{P}_{(u, \mathcal{U})} \mathbf{P}_{(u, \mathcal{U})}^T$ is a $(u \times u)$ diagonal matrix such that

$$\bar{\mathbf{P}}_{(u, \mathcal{U})}(i, j) \begin{cases} 1 & \text{if } i = j \text{ and } i \in \mathcal{U} \\ 0 & \text{otherwise} \end{cases} \text{ for } i, j \in \{0, \dots, u-1\}.$$

Then

$$\mathbf{X}_h = \bar{\mathbf{P}}_{(m_h, \mathcal{M}_h)} \mathbf{X}_h \quad \text{and} \quad \mathbf{X}_h = \mathbf{X}_h \bar{\mathbf{P}}_{(n_h, \mathcal{N}_h)}$$

hold for $h = 1, \dots, d$, from which it follows that

$$\begin{aligned} \bigotimes_{h=1}^d \mathbf{X}_h &= \bigotimes_{h=1}^d \bar{\mathbf{P}}_{(m_h, \mathcal{M}_h)} \mathbf{X}_h \bar{\mathbf{P}}_{(n_h, \mathcal{N}_h)} \\ &= \bigotimes_{h=1}^d \mathbf{P}_{(m_h, \mathcal{M}_h)} \mathbf{P}_{(m_h, \mathcal{M}_h)}^T \mathbf{X}_h \mathbf{P}_{(n_h, \mathcal{N}_h)} \mathbf{P}_{(n_h, \mathcal{N}_h)}^T \\ &= \bigotimes_{h=1}^d \mathbf{P}_{(m_h, \mathcal{M}_h)} \hat{\mathbf{X}}_h \mathbf{P}_{(n_h, \mathcal{N}_h)}^T \\ &= \bigotimes_{h=1}^d \mathbf{P}_{(m_h, \mathcal{M}_h)} \bigotimes_{h=1}^d \hat{\mathbf{X}}_h \bigotimes_{h=1}^d \mathbf{P}_{(n_h, \mathcal{N}_h)}^T \end{aligned}$$

by the compatibility of Kronecker product with matrix multiplication. $\square$

The modified shuffle algorithm is based on the next corollary.

Corollary 2. Let $\mathbf{q} = \mathbf{p} \bigotimes_{h=1}^d \mathbf{X}_h$, and let $\hat{\mathbf{X}}_h$, $\mathcal{M}_h$, and $\mathcal{N}_h$ be defined as in Lemma 5 for $h = 1, \dots, d$. Then

$$\mathbf{q}(\times_{h=1}^d \mathcal{N}_h) = \mathbf{p}(\times_{h=1}^d \mathcal{M}_h) \bigotimes_{h=1}^d \hat{\mathbf{X}}_h.$$

Proof. Using (3.3), $\mathbf{q} = \mathbf{p} \bigotimes_{h=1}^d \mathbf{X}_h$ can be written as

$$\mathbf{q} = \mathbf{p} \bigotimes_{h=1}^d \mathbf{P}_{(m_h, \mathcal{M}_h)} \bigotimes_{h=1}^d \hat{\mathbf{X}}_h \bigotimes_{h=1}^d \mathbf{P}_{(n_h, \mathcal{N}_h)}^T.$$

Then the result follows from

$$\mathbf{p}(\times_{h=1}^d \mathcal{M}_h) = \mathbf{p} \bigotimes_{h=1}^d \mathbf{P}_{(m_h, \mathcal{M}_h)}, \quad \mathbf{q}(\times_{h=1}^d \mathcal{N}_h) = \mathbf{q} \bigotimes_{h=1}^d \mathbf{P}_{(n_h, \mathcal{N}_h)},$$

and $\mathbf{P}_{(n_h, \mathcal{N}_h)}^T \mathbf{P}_{(n_h, \mathcal{N}_h)} = \mathbf{I}_{|\mathcal{N}_h|}$ for $h = 1, \dots, d$. $\square$

In Algorithm 6, flops are executed only while multiplying the vector $\hat{\mathbf{p}}$ with $\bigotimes_{h=1}^d \hat{\mathbf{X}}_h$. Hence, the cost of vector-Kronecker product multiplication using the modified shuffle algorithm amounts to

$$2 \sum_{h \in \mathcal{H}} n n z(\mathbf{X}_h) \prod_{f=1}^{h-1} |\mathcal{N}_f| \prod_{f=h+1}^d |\mathcal{M}_f| \quad (3.4)$$

Input: Vector: $\mathbf{p}$
 Rectangular Kronecker product factors: $\mathbf{X}_1, \dots, \mathbf{X}_d$
Output: Vector: $\mathbf{q} = \mathbf{p} \otimes_{h=1}^d \mathbf{X}_h$:
 1: **function** MODIFIEDSHUFFLE($\mathbf{p}, \mathbf{X}_1, \dots, \mathbf{X}_d, \mathbf{q}$)
 2: **for all** $h = 1$ to d **do**
 3: $\mathcal{M}_h = \emptyset; \mathcal{N}_h = \emptyset$;
 4: **for all** (i_h, j_h) such that $\mathbf{X}_h(i_h, j_h) > 0$ **do**
 5: $\mathcal{M}_h \leftarrow \mathcal{M}_h \cup \{i_h\}; \mathcal{N}_h \leftarrow \mathcal{N}_h \cup \{j_h\}$;
 6: **end for**
 7: Copy $\mathbf{X}_h(\mathcal{M}_h, \mathcal{N}_h)$ to $\hat{\mathbf{X}}_h$;
 8: **end for**
 9: Copy $\mathbf{p}(\times_{h=1}^d \mathcal{M}_h)$ to $\hat{\mathbf{p}}$;
 10: SHUFFLE($\hat{\mathbf{p}}, \hat{\mathbf{X}}_1, \dots, \hat{\mathbf{X}}_d, \hat{\mathbf{q}}$)
 11: Copy $\hat{\mathbf{q}}$ to $\mathbf{q}(\times_{h=1}^d \mathcal{N}_h)$;
 12: **end function**

Algorithm 6: Modified shuffle algorithm for rectangular factors

flops since $\text{nnz}(\hat{\mathbf{X}}_h) = \text{nnz}(\mathbf{X}_h)$ for $h = 1, \dots, d$. Note that the cost of Algorithm 6 in (3.4) never exceeds the cost of Algorithm 4 in (3.1) and is bounded above by $(2 |\mathcal{H}| \text{nnz}(\mathbf{X}))$ flops since $|\mathcal{M}_h| \leq \text{nnz}(\mathbf{X}_h)$ and $|\mathcal{N}_h| \leq \text{nnz}(\mathbf{X}_h)$ for $h = 1, \dots, d$.

Regarding memory requirements, each of the vectors $\mathbf{q}$ and $\mathbf{q}'$ in Algorithm 6 need to be as long as $\max_{h \in \mathcal{H}} (\prod_{f=1}^h |\mathcal{N}_f| \prod_{f=h+1}^d |\mathcal{M}_f|)$, whereas the floating-point vectors $\mathbf{z}$ and $\mathbf{z}'$ need to be at least as long as $\max_h (|\mathcal{M}_h|)$ and $\max_h (|\mathcal{N}_h|)$, respectively.

We use our running example to show how the modified shuffle algorithm works.

Example 3. (cont'd) The sets of rows and columns of the factors $\mathbf{X}_1$, $\mathbf{X}_2$, and $\mathbf{X}_3$ including at least one nonzero value are given by Algorithm 6 as

$$\mathcal{M}_1 = \{0, 1\}, \mathcal{N}_1 = \{0\}, \mathcal{M}_2 = \{1\}, \mathcal{N}_2 = \{1\}, \mathcal{M}_3 = \{0, 1\}, \text{ and } \mathcal{N}_3 = \{1, 2\}.$$

Then

$$\times_{h=1}^3 \mathcal{M}_h = \{(0, 1, 0), (0, 1, 1), (1, 1, 0), (1, 1, 1)\}, \quad \times_{h=1}^3 \mathcal{N}_h = \{(0, 1, 1), (0, 1, 2)\},$$

$$\mathbf{P}_{(m_1, \mathcal{M}_1)} = \begin{bmatrix} 1 & & \\ & 1 & \\ & & \end{bmatrix}_{3 \times 2}, \quad \mathbf{P}_{(n_1, \mathcal{N}_1)} = \begin{bmatrix} 1 \\ & \end{bmatrix}_{2 \times 1}, \quad \mathbf{P}_{(m_2, \mathcal{M}_2)} = \begin{bmatrix} & & \\ & 1 & \\ & & \end{bmatrix}_{3 \times 1},$$

$$\mathbf{P}_{(n_2, \mathcal{N}_2)} = \begin{bmatrix} & \\ & 1 \end{bmatrix}_{2 \times 1}, \quad \mathbf{P}_{(m_3, \mathcal{M}_3)} = \mathbf{I}_2, \quad \mathbf{P}_{(n_3, \mathcal{N}_3)} = \begin{bmatrix} 1 & \\ & 1 \end{bmatrix}_{3 \times 2},$$

$$\bigotimes_{h=1}^3 \mathbf{P}_{(m_h, \mathcal{M}_h)} = [\mathbf{e}_2 \quad \mathbf{e}_3 \quad \mathbf{e}_8 \quad \mathbf{e}_9]_{18 \times 4}, \quad \text{and} \quad \bigotimes_{h=1}^3 \mathbf{P}_{(n_h, \mathcal{N}_h)} = [\mathbf{e}_4 \quad \mathbf{e}_5]_{12 \times 2}.$$

Hence,

$$\begin{aligned} \hat{\mathbf{p}} &= \mathbf{p} \bigotimes_{h=1}^3 \mathbf{P}_{(m_h, \mathcal{M}_h)} = \mathbf{p}(\times_{h=1}^3 \mathcal{M}_h) \\ &= [a_2 \quad a_3 \quad a_8 \quad a_9]. \end{aligned}$$

Then $\hat{\mathbf{q}} = \hat{\mathbf{p}} \otimes_{h=1}^d \hat{\mathbf{X}}_h$ is computed using the shuffle algorithm, where

$$\hat{\mathbf{X}}_1 = \begin{bmatrix} 3 \\ 2 \end{bmatrix}, \quad \hat{\mathbf{X}}_2 = \begin{bmatrix} 2 \end{bmatrix}, \quad \text{and} \quad \hat{\mathbf{X}}_3 = \begin{bmatrix} 5 & \\ 1 & 3 \end{bmatrix}.$$

In Algorithm 4, the outer loop in line 4 is executed three times for $h = 1, 2, 3$. At the beginning of the algorithm, $\hat{\mathbf{p}}$ is copied to $\mathbf{q}$ and $\mathbf{q}' = \mathbf{q} (\hat{\mathbf{X}}_1 \otimes \mathbf{I}_2)$ is computed for $h = 1$. In this turn, $\text{nnz}(\hat{\mathbf{X}}_1) = 2$, $i_{\text{left}} = 1$, and $i_{\text{right}} = 2$. Therefore, the middle loop in line 7 and the inner loop in line 8 are executed once and twice, respectively. Then $\mathbf{q} (\hat{\mathbf{X}}_1 \otimes \mathbf{I}_2)$ is computed as

$$\mathbf{q}' = [3a_2 + 2a_8 \quad 3a_3 + 2a_9]$$

in 8 flops and $\mathbf{q}'$ is copied to $\mathbf{q}$ at the end of the first turn. In the second turn, $\mathbf{q}' = \mathbf{q} (\mathbf{I}_1 \otimes \hat{\mathbf{X}}_2 \otimes \mathbf{I}_2)$ is computed for $h = 2$. In this turn, $\text{nnz}(\hat{\mathbf{X}}_2) = 1$, $i_{\text{left}} = 1$, and $i_{\text{right}} = 2$. Therefore, the middle loop and the inner loop are executed once and twice, respectively. Then $\mathbf{q} (\mathbf{I}_1 \otimes \hat{\mathbf{X}}_2 \otimes \mathbf{I}_2)$ is computed as

$$\mathbf{q}' = [6a_2 + 4a_8 \quad 6a_3 + 4a_9]$$

in 4 flops and $\mathbf{q}'$ is copied to $\mathbf{q}$ at the end of the second turn. In the last turn, $\mathbf{q}' = \mathbf{q} (\mathbf{I}_1 \otimes \hat{\mathbf{X}}_3)$ is computed for $h = 3$. In this turn, $\text{nnz}(\hat{\mathbf{X}}_3) = 3$, $i_{\text{left}} = 1$, and $i_{\text{right}} = 1$. Therefore, the middle loop and the inner loop are each executed once. Then $\mathbf{q} (\mathbf{I}_1 \otimes \hat{\mathbf{X}}_3)$ is computed as

$$\mathbf{q}' = [30a_2 + 6a_3 + 20a_8 + 4a_9 \quad 18a_3 + 12a_9]$$

in 6 flops, and $\mathbf{q}'$ is copied to $\hat{\mathbf{q}}$ at the end of the last turn. Then the elements of $\hat{\mathbf{q}}$ are copied back to $\mathbf{q}(\times_{h=1}^3 \mathcal{N}_h)$, that is

$$\mathbf{q} = [0 \ 0 \ 0 \ 0 \ 30a_2 + 6a_3 + 20a_8 + 4a_9 \ 18a_3 + 12a_9 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0]$$

by the equation

$$\hat{\mathbf{q}} = \mathbf{q} \bigotimes_{h=1}^3 \mathbf{P}_{(n_h, \mathcal{N}_h)} = \mathbf{q}(\times_{h=1}^3 \mathcal{N}_h).$$

Hence, Algorithm 6's computation of $\mathbf{q} = \mathbf{p}\mathbf{X}$ takes altogether 18 flops. This number is smaller than 56 flops and 22 flops that shuffle and Pot-RwCl algorithms respectively take.

We remark that the proposed improvement will also be useful when the factors are relatively dense as long as some of them include zero rows or columns. In the next section, we discuss implementation issues associated with the shuffle, Pot-RwCl, and modified shuffle algorithms.

3.4 Implementation issues

We considered two benchmarks in this chapter. The first one is the implementation of the shuffle algorithm in the Nsolve package of the Abstract Petri Net Notation (APNN) toolbox [1, 14]. The pseudocode of the Nsolve implementation of the loop in line 4 of Algorithm 4 is given in [19]. Identity matrices are not stored as discussed in [19] and the multiplication is not performed for identity factors. Besides, we modified the implementation so that each matrix that is a multiple of an identity matrix is not stored either. In this implementation, each nonzero of the factor $\mathbf{X}_h$ is multiplied with the elements of the input vector and added to the appropriate elements of the output vector so that the vectors $\mathbf{z}$ and $\mathbf{z}'$ are not used and stored in the multiplication. Since we consider models with more than one Kronecker product term, the intermediate results cannot be stored in the output vector. The number of auxiliary vectors to store the intermediate results depends on the maximum of the number of nonidentity factors across all Kronecker product terms. If this number is one, then there is no intermediate

result, and therefore, no auxiliary vector is required. If the maximum number of nonidentity factors across all terms is two, then the allocation of one auxiliary vector is sufficient. Otherwise, two auxiliary vectors need to be allocated to store the intermediate results. However, it should be noted that at least two auxiliary vectors each of length $\max_i |\mathcal{R}^{(i)}|$ are required in numerical solvers. This is the case for all algorithms discussed in this chapter when they are used as kernels in numerical solvers. In the Nsolve implementation of the shuffle algorithm, the elements of the output vector are multiplied by the rate α_t associated with the Kronecker product term and added to the output vector after the multiplication of the input vector with the Kronecker product term. The cost of multiplying the output vector with α_t becomes excessive especially when the factors are sparse. In order to avoid this cost, we multiplied α_t with the nonzero elements of the first nonidentity factor and stored the modified factor. If all factors are identity matrices, the term rate is kept separately and the multiplication is obtained by multiplying the rate with the input vector.

As the second benchmark, we implemented the Pot-RwCl algorithm for rectangular factors. The pseudocode of the algorithm given in [24] for square factors includes d nested loops, but therein it is also stated that a recursive implementation is required since d is model-dependent. In order to make a fair comparison between the algorithms, we provided a non-recursive implementation with dynamically allocated arrays of size d as suggested in [24]. In the Pot-RwCl algorithm, row and column indices of the nonzero elements of Kronecker product terms need to be computed. Hence, its indexing and addressing overhead is larger than that of the shuffle algorithm. However, Pot-RwCl uses the cache more efficiently since nonzero elements of the Kronecker product of the factors are obtained consecutively and the corresponding elements of the input and output vectors tend to be closer to each other. In the shuffle algorithm, the cache is not accessed as efficiently as nonzero generation that takes place in the Pot-RwCl algorithm when the value of index h is large. Furthermore, as stated in section 3.2, Pot-RwCl does not require any intermediate vector computation, implying it does not require any additional floating-point vectors (except the two auxiliary vectors each of length $\max_i |\mathcal{R}^{(i)}|$ when it is used as a kernel in numerical solvers). To cut

down on indexing overhead in the implementation of line 8 in Algorithm 5, the input vector is multiplied with the common nonzero element when the remaining factors to be processed are identity matrices even if $h < d$.

In the implementation of the modified shuffle algorithm, we removed zero rows and columns of the nonidentity matrices and allocated two integer vectors to store the mapping between the row and column indices of the factor and the modified factor. This modification may avoid unnecessary flops only if the Kronecker product term includes more than one nonidentity factor. Besides, choosing elements of the input and output vectors yields indexing and addressing overhead in many cases. Therefore, we removed zero rows and columns of nonidentity factors in a Kronecker product term only if their counts are more than one. A naive implementation is to copy appropriate elements of the input vector to a temporary vector, multiply this temporary vector with the Kronecker product of the submatrices, and add the resulting vector to the output vector. Instead, we modified the shuffle algorithm implementation so that appropriate elements of the input vector are chosen in the first turn of the outer loop in line 4 of Algorithm 4. Similarly, the elements of the resulting vector are added to the appropriate elements of the output vector in the last turn of the outer loop in line 4 of Algorithm 4. A recursive implementation seemed to be convenient in choosing relevant elements of input and output vectors. However, in order to enable a fair comparison between the algorithms, we implemented Algorithm 6 without recursion using arrays of size d as it is done for Pot-RwCl. We preferred the same implementation even for the multiplication of the vector with Kronecker product terms in which no rows and columns are removed from the factors. Implementations of the three algorithms as discussed here are available at [50].

3.5 Numerical results

We performed numerical experiments on an Intel Core i7 Quad-Core 3.6 GHz processor with 32 GB of main memory. We considered fifteen Kronecker based MCs with given Cartesian product partitionings of their reachable state spaces.

Six of the MCs correspond to systems of stochastic chemical kinetics that are models of a gene expression [146], a toggle switch [71], an exclusive switch [112], a metabolite synthesis with one enzyme [142], a metabolite synthesis with two enzymes [142], and a repressilator [111]. One of the MCs is a queueing network model of a call center having five subsystems; it is a parallel service system known as the N-model under the threshold routing control policy proposed in [15]. Further information regarding these seven models may be obtained from [30, 55]. The remaining eight MCs are models of the Courier protocol introduced in [151], the multiserver multiqueue (MSMQ) models discussed in [5], models of manufacturing systems having Kanban control [117], and a model of token-based scheduling in a queueing network called `qh_realcontrol` [27]. These were used as benchmark problems before [22, 25, 26, 27].

Table 3.1 provides the properties associated with the Kronecker representation of the generator matrices of the MC models. The first column gives the name of the model, the second column gives the number of subsystems in the corresponding system, the third column gives the number of reachable state space partitions, the fourth column gives the number of Kronecker product terms in the representation, the fifth column gives the average number of nonzeros per row in the factors, the sixth column gives the maximum partition size of the reachable state space partitioning, the seventh column gives the number of reachable states, and the last column gives the number of nonzero off-diagonal elements in $\mathbf{Q}$ underlying the respective MC. Note that for practical reasons, in almost all cases the diagonal of $\mathbf{Q}$ is stored explicitly in Kronecker based MCs; hence, for a fairer memorywise comparison, we also do not account for that in the flat representation. Otherwise, the number of nonzeros in $\mathbf{Q}$ for each model may be found by summing the value in the last two columns of Table 3.1. The values of nonzeros in $\mathbf{Q}$ is immaterial for this work; hence, we do not provide the real parameters of the corresponding models.

We considered models of different sizes. The reachable state space sizes of the models range from about 350,000 to about 3,000,000. All models except gene expression have at least three factors in their Kronecker products. The models coming from stochastic chemical physics, `kanban_medium`, and `kanban_large` do

not have any unreachable states. Recall that all square factors is just a special case of rectangular factors, and by experimenting with these models we show that indeed there are savings that may be realized in the all square factors case as well. The maximum number of Kronecker product terms over all blocks of the generator matrix is 1,100 and appears in the N-model. It is also the N-model which has the largest number of partitions of its reachable state space with 204 partitions. The average number of nonzeros per row in the nonidentity factors of Kronecker product terms of the models ranges from 0.82 (exclusive switch) to 1.00 (gene expression, metabolite synthesis with 1 and 2 enzymes). Hence, Kronecker product factors of the models are relatively sparse for the shuffle algorithm to be efficient. If we were to compute the generator matrices corresponding to the models, we would be having $nnz(\mathbf{Q}^{off})$ nonzeros in their off-diagonal parts and we would be performing $2nnz(\mathbf{Q}^{off})$ flops when we multiply a vector of length $|\mathcal{R}|$ with $\mathbf{Q}^{off}$. Note that $2nnz(\mathbf{Q}^{off})$ flops is therefore a lower bound for the case of relatively sparse Kronecker product factors, and we should be happy to see flop counts that are close to $2nnz(\mathbf{Q}^{off})$ with vector–Kronecker product multiplication algorithms.

Table 3.1: Model properties

Model	d	p_{rt}	$term$	$dens$	$\max_i \mathcal{R}^{(i)} $	$ \mathcal{R} $	$nnz(\mathbf{Q}^{off})$
gene expression	2	1	4	1.00	1,002,001	1,002,001	4,003,000
toggle switch	3	1	8	0.87	1,085,764	1,085,764	5,418,400
exclusive switch	3	1	10	0.82	910,803	910,803	4,242,700
met. syn. 1 enz.	3	1	7	1.00	1,191,016	1,191,016	8,236,200
met. syn. 2 enz.	4	1	9	1.00	1,679,616	1,679,616	14,560,560
repressilator	4	1	12	0.90	1,191,016	1,191,016	8,764,080
N-model	5	204	1,100	0.96	39,601	489,930	2,334,358
courier_large	4	10	80	0.91	117,000	419,400	2,281,620
courier_huge	4	13	109	0.91	468,000	1,632,600	9,732,330
msmq_medium	5	15	100	0.98	26,136	358,560	2,135,160
msmq_large	5	35	250	0.99	107,653	2,945,880	19,894,875
kanban_medium	4	1	7	0.95	527,076	527,076	3,001,405
kanban_large	4	1	7	0.96	1,742,400	1,742,400	10,183,360
kanban_fail	4	8	68	0.93	291,600	2,302,911	14,313,663
qh_realcontrol	3	9	45	0.99	48,048	399,476	1,871,004

Results of numerical experiments with the models in Table 3.1 are reported in Table 3.2. For each model, we considered implementations of the three vector–Kronecker product algorithms as discussed in Section 3.4. In the second column

of the table, S, P, and MS respectively denote the shuffle, the Pot-RwCl, and the modified shuffle algorithms. Column *nnz* provides the total number of nonzeros in the explicitly stored matrices. Column *expl* provides the number of explicitly stored matrices. Column *mexpl* provides the maximum of the number of explicitly stored matrices in the Kronecker product terms. Column $E[expl]$ provides the expected number of explicitly stored matrices. Column *subm* is the number of matrices in which at least one row or one column is removed. The last three columns pertain to the performance of vector-Kronecker product multiplication. Column *aux* provides the total length of additional floating-point vectors used during multiplication. This number is determined by *mexpl* as discussed in Section 3.4. Column *flops* provides the number of flops executed in the multiplication of a vector with the off-diagonal part of $\mathbf{Q}$ in Kronecker form. The number of flops is the sum of the costs discussed in Sections 3.1, 3.2, and 3.3 over the Kronecker product terms in the blocks. The last column gives the time of the multiplication of a vector with the off-diagonal part of $\mathbf{Q}$ in Kronecker form in seconds of CPU time for 10,000 such multiplications. We remark that the time it takes to set the Kronecker representation of the infinitesimal generator matrix is negligible. The bold figures in the last three columns of Table 3.2 indicate the minimum values in those columns for the particular model.

Table 3.2: Numerical results

Model	Alg.	<i>nnz</i>	<i>expl</i>	<i>mexpl</i>	$E[expl]$	<i>subm</i>	<i>aux</i>	<i>flops</i>	Time
gene expression	S	5,000	5	2	1.25	0	1,002,001	10,010,000	301
	P	5,000	5	2	1.25	0	0	10,010,000	66
	MS	4,000	4	1	1.00	2	0	8,006,000	54
toggle switch	S	4,172	14	2	1.75	0	1,085,764	23,853,464	389
	P	4,172	14	2	1.75	0	0	15,173,600	266
	MS	2,080	4	1	0.50	12	0	10,836,800	152
exclusive switch	S	5,508	18	2	1.80	0	910,803	23,040,616	350
	P	5,508	18	2	1.80	0	0	12,427,800	300
	MS	2,200	4	1	0.40	16	0	8,485,400	169
met. syn. 1 enz.	S	1,051	10	2	1.43	0	1,191,016	23,618,072	486
	P	1,051	10	2	1.43	0	0	20,034,316	124
	MS	946	9	2	1.29	5	1,191,016	21,147,000	152
met. syn. 2 enz.	S	492	14	2	1.56	0	1,679,616	45,909,504	1,005
	P	492	14	2	1.56	0	0	34,114,642	223
	MS	422	12	2	1.33	8	1,679,616	38,646,720	278
repressilator	S	660	21	2	1.75	0	1,191,016	38,764,200	781
	P	660	21	2	1.75	0	0	23,382,112	288
	MS	312	6	1	0.50	18	0	17,528,160	180
N-model	S	30,614	682	2	0.62	0	39,601	6,437,428	71
	P	30,614	682	2	0.62	0	0	5,199,918	26
	MS	26,104	594	1	0.54	88	0	4,668,716	24

Table 3.2: Numerical results (cont'd)

Model	Alg.	nnz	$expl$	$mexpl$	$E[expl]$	$subm$	aux	$flops$	Time
courier_large	S	1,930	118	2	1.48	0	468,000	8,035,980	75
	P	1,930	118	2	1.48	0	0	5,293,496	26
	MS	1,517	57	1	0.71	76	0	4, 563, 240	23
courier_huge	S	4,540	162	2	1.49	0	1,404,000	38,095,740	553
	P	4,540	162	2	1.49	0	0	23,025,542	214
	MS	4,291	98	2	0.90	106	468,000	20, 073, 660	167
msmq_medium	S	985	125	2	1.25	0	47,916	5,193,936	34
	P	985	125	2	1.25	0	0	4,852,280	34
	MS	745	85	1	0.85	50	0	4, 270, 320	33
msmq_large	S	3,535	325	2	1.30	0	199,927	49,927,654	392
	P	3,535	325	2	1.30	0	0	45,177,891	303
	MS	3,115	265	2	1.06	150	107,653	42, 097, 454	257
kanban_medium	S	370	10	2	1.43	0	527,076	9,104,040	119
	P	370	10	2	1.43	0	0	6,996,185	49
	MS	130	4	1	0.57	6	0	6, 002, 810	41
kanban_large	S	406	10	2	1.43	0	1,742,400	30,666,240	722
	P	406	10	2	1.43	0	0	23,610,383	165
	MS	148	4	1	0.57	6	0	20, 366, 720	152
kanban_fail	S	2,916	108	2	1.59	0	364,500	50,222,268	410
	P	2,916	108	2	1.59	0	0	35, 183, 125	473
	MS	2,895	93	2	1.37	80	291,600	38,558,718	415
qh_realcontrol	S	2,331	63	2	1.40	0	70,070	5,199,112	48
	P	2,331	63	2	1.40	0	0	4, 618, 858	26
	MS	2,331	63	2	1.40	32	48,048	4,866,972	30

For each model, number of flops in vector-Kronecker product multiplication decreases when the shuffle algorithm is modified. There are two reasons for this decrease. First, unnecessary flops in the loop in line 4, for some fixed h , in Algorithm 4 are avoided. Second, some factors become multiples of identity matrices once the removal of zero rows and columns take place, and the loop in line 4 of Algorithm 4 is avoided for such factors. The improvement in number of flops is smallest at 6% for `qh_realcontrol`, the only model where the number of explicitly stored matrices does not decrease after the modification. In the other models, the improvement in the number of flops ranges from 10% (metabolite synthesis with 1 enzyme) to 63% (exclusive switch) with an average of 31% over fifteen models. In all models but six (metabolite synthesis with 1 and 2 enzymes, `courier_huge`, `msmq_large`, `kanban_fail`, and `qh_realcontrol`), the modified shuffle algorithm performs $2nnz(\mathbf{Q}^{off})$ flops, which is the lower bound. We remark that in each of these nine models, there is no term with more than one explicitly stored matrix (see the values in column $mexpl$). Moreover, in `courier_huge` and `msmq_large`, modified shuffle yields the minimum number of flops among the three algorithms. Hence, modified shuffle is a winner in terms of number of flops in

eleven models. In the nine models where modified shuffle performs $2nnz(\mathbf{Q}^{off})$ flops, it also does not require additional memory. Besides, the modification also decreases the memory allocated for auxiliary vectors over that of shuffle in some other models such as `courier_huge`, `msmq_large`, `kanban_fail`, and `qh_realcontrol`. As expected, the number of flops executed by the Pot-RwCl algorithm is not more than that of the shuffle algorithm since the factors of Kronecker product terms in all models are relatively sparse. Pot-RwCl yields a smaller number of flops than modified shuffle only in four models (metabolite synthesis with 1 and 2 enzymes, `kanban_fail`, and `qh_realcontrol`). It is the value of $E[expl]$, which is relatively larger than 1, that causes modified shuffle to suffer in these four models. The number of flops for Pot-RwCl and shuffle are the same in gene expression. The equality is due to the choice of integer model parameters. For N-model and `msmq_large`, the difference in number of flops between Pot-RwCl and modified shuffle is not larger than 10%. In the other models, modified shuffle yields improvements in number of flops ranging from 12% to 36% over Pot-RwCl. Note that it is not feasible to use Pot-RwCl in models having relatively denser factors for which shuffle starts becoming quite efficient. Therefore, the results on these fifteen models are in some sense indicative of the best Pot-RwCl can do.

The improvement in the number of flops obtained with modified shuffle in (3.4) can be predetermined by comparing it with (3.1) and (3.2). This is also true for the improvement in memory, if there is any. But, the improvement in time depends on the particular factors in the Kronecker product terms. Besides the number of flops, the time that the algorithms take also depends on the overhead of indexing and addressing as well as the access pattern to the input and output vectors. Time per flop (i.e., Time/flops) seems to be a good measure to understand the overhead and cache usage of algorithms. Time per flop of shuffle and Pot-RwCl are almost the same for `toggle switch` and `msmq_medium`. Shuffle is better than Pot-RwCl in time per flop for `exclusive switch` and `kanban_fail`. Shuffle and Pot-RwCl take almost the same time for `msmq_medium` and shuffle takes smaller time than Pot-RwCl only for `kanban_fail`. In the other models, Pot-RwCl is timewise better than shuffle. This indicates that the cost due to indexing, addressing and access pattern is smaller than the gain obtained

from the decrease in the number of flops. On the other hand, time per flop of modified shuffle is larger than that of shuffle in exclusive switch, msmq-medium, and kanban_fail. Except exclusive switch, the decrease in the number of flops does not compensate for the overhead in the implementation. In other models, modified shuffle yields an improvement of up to 82% (gene expression) in time over shuffle. Time per flops of modified shuffle and Pot-RwCl are close to each other since these algorithms are implemented in a similar manner as discussed in Section 3.4. Modified shuffle has an overhead due to the mapping between rows and columns of the original and modified factors. However, it requires a smaller number of index operations in some models. The effect of these together seem to cancel, and the difference between time per flop values of Pot-RwCl and modified shuffle ends up being relatively small.

Chapter 4

Compact vector representation of solution vectors

The Kronecker-based approach allows a compact representation of the generator matrix; however, the memory allocated for the vectors is still proportional to the size of the reachable state space, and this size increases rapidly with the number of dimensions. The current chapter attempts to reduce the amount of memory allocated to solution vectors in Kronecker-based Markovian analysis by using the hierarchical Tucker decomposition (HTD) [81, 84, 104, 105]. HTD is originally conceived in the context of providing a compact approximate representation for dense multi-dimensional data [83] in a manner similar to the tensor-train decomposition [123], but is more suitable to our requirements in that the decomposition is available through a tree data structure with logarithmic depth in the number of dimensions. Both decompositions have the special feature of possessing approximation errors that can be user controlled, and hence, approximations accurate to machine precision are computable using them. Clearly, with such decompositions it is always possible to trade quality of approximation for compactness of representation, and how compact the solution vector in HTD format remains throughout the solution process is an interesting question to investigate. The tensor train decomposition has been applied in [103] to approximate the solution vector for models where the product space is reachable using an alternating least

squares approach. HTD has, to the best of our knowledge, not been applied to structured Markov chains yet.

In this chapter, we show that a compact solution vector in HTD format can be multiplied with a sum of Kronecker products to yield another compact solution vector in HTD format. In doing this, we note that the multiplication of the compact solution vector in HTD format with a Kronecker product term does not increase the memory requirements of the compact vector, but the addition of two compact vectors does, which necessitates some kind of truncation, hence, approximation, to be introduced to the addition operation. Then, starting from an initial solution, the compact vector in HTD format is iteratively multiplied with the uniformized generator matrix of a given MC in Kronecker form until a predetermined stopping criterion is met. Indeed, we are interested in observing how the memory requirements of the compact solution vector in HTD format changes over the course of iterations due to the sequence of multiply, add, and truncate operations in each iteration, together with the average time it takes to perform the iteration and the influence of the approximation error on the quality of the solution. The same numerical experiment is performed with a flat solution vector as long as the reachable state space size using the modified shuffle algorithm. The two approaches are compared for their memory and timing requirements, leading us to the conclusion that compact vectors in HTD format become relatively more memory efficient as the number of dimensions increases.

We remark that compact representations for solution vectors in Markovian analysis have also been considered from the perspective of binary decision diagrams [31, 108]. The proposed compact structures therein have not been timewise competitive, whereas the approach investigated in this chapter seems to be a step forward. In the next section, we provide background information on HTD and the related algorithms that are to be used in our Kronecker setting. In Section 4.2, we discuss implementation issues associated with using HTD within the NSolve package of the Abstract Petri Net Notation (APNN) toolbox [1, 14]. In the final section, we present results of numerical experiments on two different problems of varying sizes and having transitions that take place at different time scales.

4.1 Compact vectors in Kronecker setting

To simplify the discussion and the notation, we consider the multiplication of a single block of $\mathbf{Q}$ from the left with a (sub)vector, and therefore, omit the indices (l, l') and write the index j associated with the transition as a superscript in parentheses above the matrices forming the block. Hence, we concentrate on the operation

$$\mathbf{y}^T := \mathbf{x}^T \sum_{j=1}^J \bigotimes_{h=1}^d \mathbf{S}_h^{(j)},$$

where $\mathbf{S}_h^{(j)}$ is a $(m_h \times n_h)$ matrix, implying $\bigotimes_{h=1}^d \mathbf{S}_h^{(j)}$ is a $(\prod_{h=1}^d m_h \times \prod_{h=1}^d n_h)$ matrix, and $\mathbf{x}$ is a $(\prod_{h=1}^d m_h \times 1)$ vector. J is equal to the number of terms in the sum, i.e., $|\mathcal{T}^{(l, l')}|$ if we consider block (l, l') . Observe that this is the operation that takes place when each block of a block matrix in Kronecker form such as $\mathbf{Q}$ gets multiplied on the left by an iteration subvector. In fact, the same subvector multiplies all blocks in a row of the matrix in Kronecker form.

To be consistent with the literature, we consider in the following multiplications of Kronecker products $\bigotimes_{h=1}^d \mathbf{A}_h^{(j)}$ with column vector $\mathbf{x}$ and their summation in the usual matrix-vector form

$$\mathbf{y} := \sum_{j=1}^J \left(\bigotimes_{h=1}^d \mathbf{A}_h^{(j)} \right) \mathbf{x},$$

where $\mathbf{A}_h^{(j)}$ is the transpose of $\mathbf{S}_h^{(j)}$ and of size $(n_h \times m_h)$. In particular, we are interested in its implementation as

$$\mathbf{y}^{(1)} := \mathbf{0}, \quad \mathbf{x}^{(j)} := \left(\bigotimes_{h=1}^d \mathbf{A}_h^{(j)} \right) \mathbf{x}, \quad \mathbf{y}^{(j+1)} := \mathbf{y}^{(j)} + \mathbf{x}^{(j)} \quad \text{for } j = 1, \dots, J,$$

and $\mathbf{y} := \mathbf{y}^{(J+1)}$, where $\mathbf{0}$ is a column vector of 0's. Now, we turn to the HTD format.

4.1.1 HTD format

Assuming without loss of generality that d is a power of 2, the $(\prod_{h=1}^d m_h \times 1)$ vector $\mathbf{x}$ in (orthogonalized) HTD format can be expressed as

$$\mathbf{x} = (\mathbf{U}_1 \otimes \cdots \otimes \mathbf{U}_d) \mathbf{c},$$

where $\mathbf{U}_h$ for $h = 1, \dots, d$ are $(m_h \times r_h)$ orthogonal basis matrices for the different dimensions in the model and

$$\mathbf{c} = (\mathbf{B}_{1,2} \otimes \cdots \otimes \mathbf{B}_{d-1,d}) \cdots (\mathbf{B}_{1,\dots,d/2} \otimes \mathbf{B}_{d/2+1,\dots,d}) \mathbf{B}_{1,\dots,d}$$

is a $(\prod_{h=1}^d r_h \times 1)$ vector in the form of a product of $\log_2 d$ matrices each of which except the last is a Kronecker product of a number of transfer matrices $\mathbf{B}_t$ related to each other as in the full binary tree of Figure 4.1. The transfer matrix $\mathbf{B}_t$ is of size $(r_{t_l} r_{t_r} \times r_t)$ with the node index t defined as $t := t_l, t_r$, and $r_{1,\dots,d} = 1$ since $\mathbf{B}_{1,\dots,d}$ is at the root of the tree [104, pp. 5-6].

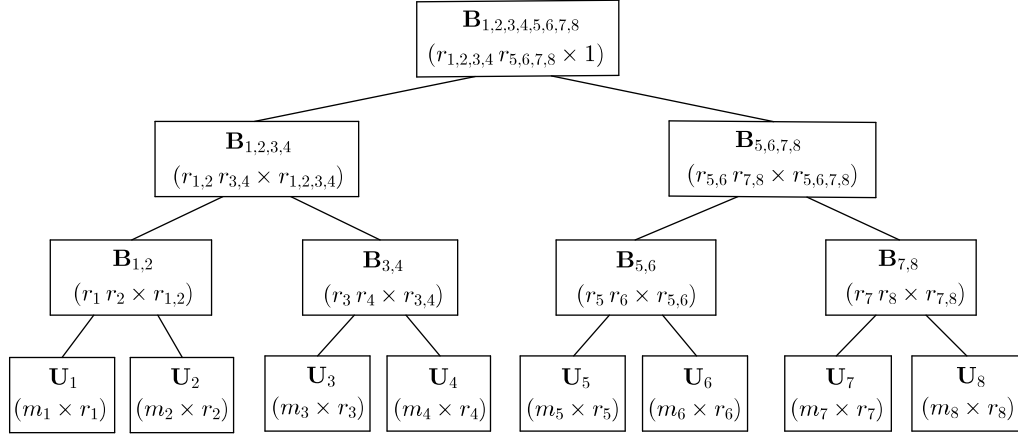


Figure 4.1: Matrices forming $\mathbf{x}$ in HTD format for $d = 8$.

The $(d - 1)$ intermediate nodes of the binary tree in Figure 4.1 store the transfer matrices $\mathbf{B}_t$ and its leaves store the basis matrices $\mathbf{U}_h$ so that each intermediate node has two children. In orthogonalized HTD format of $\mathbf{x}$, one can also conceive of orthogonal basis matrices $\mathbf{U}_t = (\mathbf{U}_{t_l} \otimes \mathbf{U}_{t_r}) \mathbf{B}_t$, at intermediate nodes with r_t columns that relate the orthogonal basis matrices $\mathbf{U}_{t_l}$ and $\mathbf{U}_{t_r}$ for the two children of transfer matrix $\mathbf{B}_t$ with the transfer matrix itself. In fact,

the orthogonal matrix $\mathbf{U}_t$ has in its columns the singular vectors associated with the largest r_t singular values [77, pp. 76-79] of the matrix obtained by taking index t as row index, the remaining indices in order as column index of the d -dimensional data at hand (i.e., with a slight abuse of notation, $\mathbf{x}(t, \{1, \dots, d\} \setminus \{t\})$). Hence, we have the concepts of “hierarchy of matricizations” and “higher-order singular value decomposition (HOSVD)” [109], and r_t is the rank of the truncated HOSVD. More detailed information regarding this can be found in [83, 104]. We remark that $\mathbf{B}_t$ may also be viewed as a 3-dimensional array of size $(r_{t_l} \times r_{t_r} \times r_t)$ having as many indices in each of its three dimensions as the number of columns in the matrices in its two children and itself, respectively. The number of transfer matrices in the l th factor forming $\mathbf{c}$ is the Kronecker product of $2^{\log_2 d - l}$ transfer matrices for $l = 1, \dots, \log_2 d - 1$. In fact, $\mathbf{c}$ is a product of Kronecker products, and so is $\mathbf{x}$, but neither has to be formed explicitly.

When d is not a power of 2, it is still useful to keep the tree in a balanced form, for instance, as in Figure 4.2 for which

$$\mathbf{x} = (((\mathbf{U}_1 \otimes \mathbf{U}_2)\mathbf{B}_{1,2}) \otimes \mathbf{U}_3 \otimes \mathbf{U}_4 \otimes \mathbf{U}_5)(\mathbf{B}_{1,2,3} \otimes \mathbf{B}_{4,5})\mathbf{B}_{1,2,3,4,5}.$$

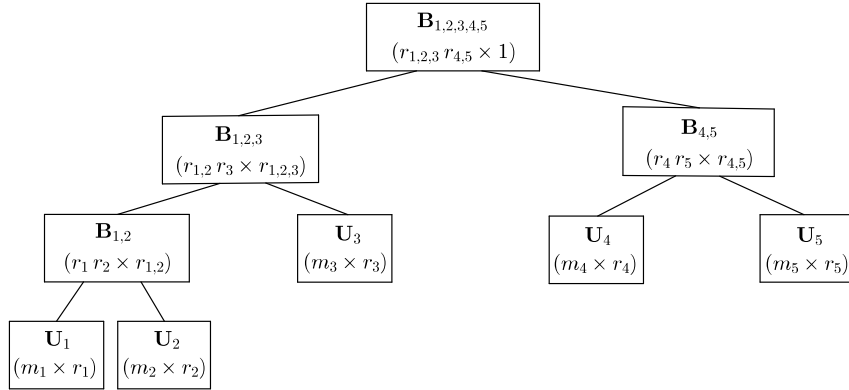


Figure 4.2: Matrices forming $\mathbf{x}$ in HTD format for $d = 5$.

Assuming that $r_{\max} = \max_t(r_t)$ and $m_{\max} = \max(m_1, \dots, m_d)$, memory requirement for matrices in the binary tree associated with HTD format is bounded by $dm_{\max}r_{\max}$ at the leaves, $r_{\max}^2$ at the root, and $(d-2)r_{\max}^3$ at other intermediate

nodes, thus, totally $dm_{\max}r_{\max} + (d-2)r_{\max}^3 + r_{\max}^2$. In the next subsection, we show how a particular rank-1 vector can be represented in HTD format.

4.1.2 Uniform distribution in HTD format

Let $\mathbf{x} = \mathbf{e}/m$ be the $(m \times 1)$ uniform distribution vector, where $m = \prod_{h=1}^d m_h$. Then $\mathbf{x}$ may be represented in HTD format with all matrices having rank-1 for which the basis matrices given by $\mathbf{U}_h = \mathbf{e}/\sqrt{m_h}$ are of size $(m_h \times 1)$ for $h = 1, \dots, d$ and the transfer matrices given by

$$\mathbf{B}_t = \begin{cases} (\prod_{h=1}^d \sqrt{m_h})/m & \text{if } t \text{ corresponds to root} \\ 1 & \text{otherwise} \end{cases}$$

are (1×1) . Note that memory taken up by flat representation of $\mathbf{x}$ is m nonzeros, whereas that with HTD format is $d-1 + \sum_{h=1}^d m_h$ nonzeros since the $(d-1)$ transfer matrices are all scalars equal to 1 except the one corresponding to the root. In passing to the multiplication of a compact vector with a Kronecker product, we remark that each basis matrix $\mathbf{U}_h$ for the uniform distribution has only a single column and that column is unit 2-norm, implying all $\mathbf{U}_h$ are orthogonal.

4.1.3 Multiplication of vector in HTD format with Kronecker product

Assuming that $\mathbf{x}$ is in HTD format with orthogonal basis matrices $\mathbf{U}_h$ and transfer matrices $\mathbf{B}_t$ forming vector $\mathbf{c}$, the operation

$$\mathbf{x}^{(j)} := \left(\bigotimes_{h=1}^d \mathbf{A}_h^{(j)} \right) \mathbf{x} \text{ is equivalent to performing } \mathbf{x}^{(j)} := \left(\bigotimes_{h=1}^d \mathbf{A}_h^{(j)} \mathbf{U}_h \right) \mathbf{c}$$

since $\mathbf{x} = (\bigotimes_{h=1}^d \mathbf{U}_h) \mathbf{c}$. Hence, the only thing that needs to be done to carry out the computation of $\mathbf{x}^{(j)}$ in HTD format is to multiply the $(n_h \times m_h)$ Kronecker factor $\mathbf{A}_h^{(j)}$ with the corresponding $(m_h \times r_h)$ orthogonal basis matrix $\mathbf{U}_h$ for $h = 1, \dots, d$. Clearly, the $(n_h \times r_h)$ product matrix $\mathbf{A}_h^{(j)} \mathbf{U}_h$ need not be orthogonal. But this does not pose much of a problem, since $\mathbf{x}^{(j)}$ can be transformed

into orthogonalized HTD format if the need arises by computing the QR decomposition [77, pp. 246-250] of $\mathbf{A}_h^{(j)} \mathbf{U}_h = \tilde{\mathbf{U}}_h \mathbf{R}_h$ for $h = 1, \dots, d$, propagating the triangular factors $\mathbf{R}_h$ into the transfer matrices, and orthogonalizing the updated transfer matrices at intermediate nodes in a similar manner up to the root as in Algorithm 1 in [104, p. 12]. However, the situation is not as good for the addition of two compact vectors.

4.1.4 Addition of two vectors in HTD format and truncation

Addition of two matrices $\mathbf{Y}$ and $\mathbf{X}$ with given singular value decompositions (SVDs) [77, pp. 76-79]

$$\mathbf{Y} = \mathbf{U}_Y \Sigma_Y \mathbf{V}_Y^T \quad \text{and} \quad \mathbf{X} = \mathbf{U}_X \Sigma_X \mathbf{V}_X^T$$

results in

$$\mathbf{Y} + \mathbf{X} := (\mathbf{U}_Y \ \mathbf{U}_X) \begin{pmatrix} \Sigma_Y & \\ & \Sigma_X \end{pmatrix} (\mathbf{V}_Y \ \mathbf{V}_X)^T.$$

Here, Σ_Y, Σ_X are diagonal matrices of singular values, whereas $\mathbf{U}_Y, \mathbf{U}_X$ and $\mathbf{V}_Y, \mathbf{V}_X$ are orthogonal matrices of left and right singular (row) vectors associated with matrices $\mathbf{Y}, \mathbf{X}$, respectively. SVD is a rank revealing factorization in that the number of nonzero singular values of a matrix corresponds to its column rank. This implies that the sum $(\mathbf{Y} + \mathbf{X})$ has a rank equal to the sum of the ranks of the two matrices that are added.

The situation for the sum $\mathbf{y}^{(j+1)}$ of the two vectors $\mathbf{y}^{(j)}$ and $\mathbf{x}^{(j)}$ in HTD format is no different if one replaces the SVD with HOSVD. This is conveniently illustrated for $d = 4$ by Figure 5 in [104, p. 11]. For the following steps performing the addition and representing the resulting vector in HTD format, we exploit the algorithms presented in [104]. Among three alternative approaches that have been investigated therein for computing $\mathbf{y}$, the best seems to be to multiply, add and then truncate J times as demonstrated in Figure 11 of [104]. This approach is coded in Algorithm 7 of [104, p. 23] which works by calling Algorithm 3 that

takes care of the reduced Gramians computations of a compact vector in non-orthogonalized HTD format. Recall that the compact vector $\mathbf{x}^{(j)}$ obtained after multiplication does not need to be in orthogonal HTD format even though $\mathbf{x}$ might have been. Once Algorithm 3 is executed, Algorithm 7 takes over and computes the truncated HOSVD for the sum of two vectors $\mathbf{y}^{(j)}$ and $\mathbf{x}^{(j)}$ in HTD format without initial orthogonalization. The output $\mathbf{y}^{(j+1)}$ of Algorithm 7 is a truncated compact vector in orthogonalized HTD format and this operation is repeated J times until $\mathbf{y}$ is obtained. The number of flops executed by Algorithm 7 is $O(dJ^2r_{max}^2(n_{max} + r_{max}^2 + Jr_{max}))$, where $n_{max} = \max(n_1, \dots, n_d)$. The significance of this algorithm is that one can impose an accuracy of ϵ on the truncated HOSVD by choosing rank r_t in node t based on dropping the smallest singular values whose squared sum is less than or equal to $\epsilon^2/(2d-3)$ [104, pp. 18-19]. This is a very nice result but also implies that the truncation leads to an approximate solution vector.

4.1.5 Computing the 2-norm of a vector in HTD format

Normally, it is more relevant to compute the maximum (i.e., infinity) norm of a solution vector in probabilistic analysis even though all norms are known to be equivalent [77, pp. 68-70]. However, the computation of the maximum value (in magnitude) of the elements of a compact vector requires being able to know which indexed value is the largest and also its value, which seems to be costly for a compact vector in HTD format. Therefore, we consider the computation of the 2-norm of vector $\mathbf{y}$ given by $\|\mathbf{y}\|_2 = \sqrt{\mathbf{y}^T \mathbf{y}}$.

Fortunately, $\|\mathbf{y}\|_2$ can be obtained using Algorithm 2 in [104, p. 14], which computes inner products of two compact vectors in HTD format. Here, the only difference is that the two vectors are the same vector $\mathbf{y}$. The algorithm starts from the leaves of the binary tree and moves towards the root, requiring the same sequence of operations in the first part of the computation of reduced Gramians in Algorithm 3 in [104, p. 17]. But, this has already been discussed in the previous subsection.

Now, we can move to implementation issues regarding compact solution vectors in HTD format for Kronecker-based Markovian representations.

4.2 Implementation issues

The implementation is done within the NSolve package of the APNN Toolbox [1, 14]. The binary tree data structure accompanying the HTD format is allocated at the outset depending on the value of d . It is stored in the form of an array of tree nodes from root to leaves level by level so that accessing the children of a parent node or the parent of a child node becomes relatively easy. In a tree node t , there are pointers to matrices $\mathbf{U}_t$ for leaves and $\mathbf{B}_t$ for intermediate nodes which we have seen and accounted for before, but also pointers to matrices $\mathbf{R}_t$ and, as we explain shortly, (2×2) block matrices $\mathbf{M}_t$ and $\mathbf{G}_t$ for each node. Since we expect solution vectors to be dense, the matrices in the compact representation are stored as full matrices including those corresponding to the blocks of $\mathbf{M}_t$ and $\mathbf{G}_t$. The nonzero elements of the full matrices are kept in a one-dimensional real array so that relevant LAPACK methods available at [2] can be called without having to copy vectors. We choose to store transposes of the matrices representing the compact solution vector in row sparse format (meaning they are stored by columns) so that relevant LAPACK methods can be called without having to transpose the input matrices.

The multiplication of the sparse Kronecker factors $\mathbf{A}_h^{(j)}$ with the orthogonal basis matrices $\mathbf{U}_h$ in $\mathbf{x}^{(j)} := \left(\bigotimes_{h=1}^d \mathbf{A}_h^{(j)} \mathbf{U}_h \right) \mathbf{c}$ is implemented using straightforward sparse matrix-vector multiplication. After the compact vector $\mathbf{x}^{(j)}$ is computed, the tree nodes of $\mathbf{y}^{(j)}$ are visited and its respective fields are updated so that we have $\mathbf{y}^{(j+1)}$ at hand. Efficient computation of the reduced Gramian matrices $\mathbf{G}_t$ as in Algorithm 3 of [104, p. 17] for $\mathbf{y}^{(j+1)}$ requires exploiting the block structure of the new transfer matrices $\mathbf{B}_t$ whose blocks are already available in the corresponding tree nodes of $\mathbf{y}^{(j+1)}$ after the addition operation. Clearly, there is no need to generate block matrices (or a cubic blocks as in Figure 5 of [104, p. 11]) with these blocks explicitly. We prefer to store $\mathbf{M}_t$ and $\mathbf{G}_t$ as (2×2)

block matrices because of the add a term and then truncate approach followed. Let us next elaborate on this.

Assuming that $r_t(\mathbf{y}^{(j)})$ and $r_t(\mathbf{x}^{(j)})$ denote the ranks of matrices in compact representations of the two vectors that are summed up in node t , $\mathbf{M}_t$ and $\mathbf{G}_t$ become $(r_t(\mathbf{y}^{(j)})r_t(\mathbf{x}^{(j)}) \times r_t(\mathbf{y}^{(j)})r_t(\mathbf{x}^{(j)}))$ matrices, where the first diagonal block is $(r_t(\mathbf{y}^{(j)}) \times r_t(\mathbf{y}^{(j)}))$ and the second diagonal block is $(r_t(\mathbf{x}^{(j)}) \times r_t(\mathbf{x}^{(j)}))$. Then the computation $\mathbf{M}_t := \mathbf{U}_t^T \mathbf{U}_t$ for leaf nodes can be formulated in (2×2) block manner as

$$\mathbf{M}_t^{(i,i')} := (\mathbf{U}_t^{(i)})^T (\mathbf{U}_t^{(i')}) \quad \text{for } i, i' = 1, 2,$$

where $\mathbf{U}_t^{(1)}$ and $\mathbf{U}_t^{(2)}$ denote basis matrices of $\mathbf{y}^{(j)}$ and $\mathbf{x}^{(j)}$ at leaf node t , respectively. This computation requires multiplying two full matrices for which the DGEMM routine of LAPACK may be used. On the other hand, the computation $\mathbf{M}_t := \mathbf{B}_t^T (\mathbf{M}_{t_l} \otimes \mathbf{M}_{t_r}) \mathbf{B}_t$ for intermediate nodes can be formulated from the bottom of the tree to the root in (2×2) block manner as

$$\mathbf{M}_t^{(i,i')} := (\mathbf{B}_t^{(i)})^T (\mathbf{M}_{t_l}^{(i,i')} \otimes \mathbf{M}_{t_r}^{(i,i')}) (\mathbf{B}_t^{(i')}) \quad \text{for } i, i' = 1, 2,$$

where $\mathbf{B}_t^{(1)}$ and $\mathbf{B}_t^{(2)}$ denote transfer matrices of $\mathbf{y}^{(j)}$ and $\mathbf{x}^{(j)}$ at node t , respectively.

Similarly, we have reduced Gramian computations, but in opposite direction from root to leaves, that can be formulated in (2×2) block manner for $i, i' = 1, 2$ as $\mathbf{G}_t^{(i,i')} := 1$ when t corresponds to root; otherwise,

$$\mathbf{G}_{t_l}^{(i,i')} := (\mathbf{B}_{t,2,3}^{(i)})^T (\mathbf{M}_{t_r}^{(i,i')} \otimes \mathbf{G}_t^{(i,i')}) \mathbf{B}_{t,2,3}^{(i')}$$

and

$$\mathbf{G}_{t_r}^{(i,i')} := (\mathbf{B}_{t,1,3}^{(i)})^T (\mathbf{M}_{t_l}^{(i,i')} \otimes \mathbf{G}_t^{(i,i')}) \mathbf{B}_{t,1,3}^{(i')},$$

where $\mathbf{B}_{t,2,3}^{(1)}$ and $\mathbf{B}_{t,1,3}^{(1)}$ are transfer matrices $\mathbf{B}_t^{(1)}$ of $\mathbf{y}^{(j)}$ organized respectively as $(r_{t_r}(\mathbf{y}^{(j)})r_t(\mathbf{y}^{(j)}) \times r_{t_l}(\mathbf{y}^{(j)}))$ and $(r_{t_l}(\mathbf{y}^{(j)})r_t(\mathbf{y}^{(j)}) \times r_{t_r}(\mathbf{y}^{(j)}))$ matrices and $\mathbf{B}_{t,2,3}^{(2)}$ and $\mathbf{B}_{t,1,3}^{(2)}$ are transfer matrices $\mathbf{B}_t^{(2)}$ of $\mathbf{x}^{(j)}$ organized respectively as $(r_{t_r}(\mathbf{x}^{(j)})r_t(\mathbf{x}^{(j)}) \times r_{t_l}(\mathbf{x}^{(j)}))$ and $(r_{t_l}(\mathbf{x}^{(j)})r_t(\mathbf{x}^{(j)}) \times r_{t_r}(\mathbf{x}^{(j)}))$ matrices. Such matrices are called matricizations of the given matrix (in this case, the transfer matrix

$\mathbf{B}_t^{(1)}$ or $\mathbf{B}_t^{(2)}$ along specific dimensions), and therefore, represent different organizations of the same data. We remark that the off-diagonal blocks of $\mathbf{M}_t$ and $\mathbf{G}_t$ respectively satisfy the relationships $\mathbf{M}_t^{(i,i')} = (\mathbf{M}_t^{(i',i)})^T$ and $\mathbf{G}_t^{(i,i')} = (\mathbf{G}_t^{(i',i)})^T$. Therefore, only one off-diagonal block for these two matrices in each node needs to be computed. The computation of the three blocks of $\mathbf{M}_t$ and $\mathbf{G}_t$ requires multiplications using **DGEMM** with matricizations and contraction of multi-dimensional data involving $\mathbf{B}_t^{(i)}$ matrices for $i = 1, 2$ as discussed in [104, pp. 9-10,12-13]. We use two auxiliary vectors of length $\max_{t_l, t_r, t}(r_{t_l} r_{t_r} r_t)$ to implement these operations. The disadvantage of not storing $\mathbf{M}_t$ and $\mathbf{G}_t$ as (2×2) block matrices is that longer auxiliary vectors would need to be allocated.

Truncation of a compact vector requires QR and singular value decompositions [77, pp. 76-79, 246-250] as in Algorithm 7 of [104, p. 23] to be performed. In order to compute these decompositions, **DGEQRF** and **DGESDD** routines of LAPACK are used. Since we expect input matrices to be dense, we do not call routines expecting sparse matrices. For a leaf node t , the $(m_t \times (r_t(\mathbf{y}^{(j)}) + r_t(\mathbf{x}^{(j)})))$ input matrix $\mathbf{U}_t$ maybe obtained by concatenating the matrices $\mathbf{U}_t^{(1)}$ and $\mathbf{U}_t^{(2)}$ corresponding to $\mathbf{y}^{(j)}$ and $\mathbf{x}^{(j)}$, respectively. Since the input matrix is also an output matrix, the upper-triangular factor $\mathbf{R}_t$ of the QR decomposition is returned from **DGEQRF** in the upper-triangular part of the input matrix in which the lower-triangular part has the Householder reflections amounting to the orthogonal factor $\mathbf{Q}_t$. After $\mathbf{R}_t$ is obtained, $\mathbf{R}_t \mathbf{G}_t \mathbf{R}_t^T$ needs to be formed. To this end, we first transform the block matrix $\mathbf{G}_t$ to a dense matrix (with a single block) and multiply this new matrix held as a one-dimensional array from left and right using the **DTRMM** routine of LAPACK. Note that **DTRMM** does not accept a trapezoid $\mathbf{R}_t$; however, this case can be handled by multiplying triangular and rectangular parts of $\mathbf{R}_t$ separately using **DTRMM** and **DGEMM**. Hence, there is no need to copy the output of **DGEQRF** to another matrix including $\mathbf{R}_t$. Once $\mathbf{R}_t \mathbf{G}_t \mathbf{R}_t^T$ is formed, it needs to be decomposed for its singular values and vectors. To this end, we prefer to use the **DGESDD** routine over the **DGESVD** routine since it is said to be faster [2]. We remark that this routine computes singular values through the symmetric eigenvalue decomposition, and the singular vectors are truncated at a certain number or possibly by omitting some corresponding to the smaller singular values based

on an error tolerance. $\mathbf{S}_t$ ends up being the matrix holding the r_t singular vectors. Then the orthogonal basis matrix $\mathbf{U}_t = \mathbf{Q}_t \mathbf{S}_t$ is computed using the DORMQR routine. In order to avoid storing $\mathbf{S}_t$, we prefer to update $\mathbf{R}_t$ with $\mathbf{S}_t^T$ as in the `htucker` package [105].

The same sequence of operations are carried out level by level from the parents of the leaves to the top of the tree excluding the root. The product $\mathbf{S}_t^T \mathbf{R}_t$ is computed using DTRMM (also possibly with an additional call to DGEMM when $\mathbf{R}_t$ is trapezoid) and stored in the matrix that was allocated for $\mathbf{R}_t$. Note that $\mathbf{S}_t^T \mathbf{R}_t = (\mathbf{F}_t^{(1)} \mathbf{F}_t^{(2)})$ is an $(r_t \times (r_t(\mathbf{y}^{(j)}) + r_t(\mathbf{x}^{(j)})))$ matrix with the two blocks $\mathbf{F}_t^{(l)}$ for $l = 1, 2$, where r_t is the rank of node t after truncation. Then for a non-leaf node t , the QR factorization of $\sum_{i=1}^2 (\mathbf{F}_{t_l}^{(i)} \otimes \mathbf{F}_{t_r}^{(i)}) \mathbf{B}_t^{(i)}$ needs to be computed. This computation requires multiplications using DGEMM with matricizations of multi-dimensional data involving $\mathbf{B}_t^{(i)}$ matrices for $i = 1, 2$ as discussed in [104, pp. 9-10]. Finally, the transfer matrix $\mathbf{B}_t = \mathbf{Q}_t \mathbf{S}_t$ is computed using DORMQR.

4.3 Numerical results

In this section, we consider two example models that have been used as benchmarks in [28]. The first is an availability model with d subsystems in which different time scales occur. Each subsystem models a processing node with 2 processors, one acting as a cold spare, a bus and two memory modules. The processing node is available as long as one processor can access one memory module via the bus. Time to failure is exponentially distributed with rate 5×10^{-4} for processors, 4×10^{-4} for buses and 10^{-4} for memory modules. Components are repaired by a global repair facility with preemptive priority such that components from subsystem 1 have the highest priority and components from subsystem d have the least priority. The repair times of components are exponentially distributed. The repair rates of a processor, a bus, and a memory from subsystem 1 are given respectively as 1, 2, and 4. The same rates for other subsystems are given respectively as 0.1, 0.2, and 0.4. For this model, the reachable state space is equal to the product state space and contains 12^d states. We consider availability

Table 4.1: Properties of availability and polling models

d	L	Availability	L	Polling
		$ \mathcal{R} $		$ \mathcal{R} $
3	1	1,728	6	25,443
4	1	20,736	10	479,886
5	1	248,832	15	8,065,860
6	1	2,985,984	21	125,839,395
7	1	35,831,808	28	1,863,521,121
8	1	429,981,696		

models with $d = 3, 4, 5, 6, 7, 8$.

The second example is a model of a polling system of two servers serving customers from d finite capacity queues, which are cyclically visited by the servers. Customers arrive to the system according to a Poisson process with rate 1.5 and are distributed with queue specific probabilities among the queues each of which is assumed to have a capacity of 10. If a server visits a nonempty queue, it serves one customer and then travels to the next queue. On the other hand, a server arriving at an empty queue, skips the queue and travels to the next queue. Service and travelling times of servers are exponentially distributed respectively with rates 1 and 10. We further assume that there can be at most one customer being served at a time in each queue and at most one server travelling at a time from each queue to the next queue. Each subsystem in the model describes one queue, and the L partitions of the reachable state space for this model are defined according to the number of servers serving customers at a queue or travelling to the next queue. For each subsystem we obtain 62 states partitioned into 3 subsets. The reachable state space of the complete model has $L = \binom{d+1}{2}$ partitions, and we consider polling system models with $d = 3, 4, 5, 6, 7$.

The goal of this chapter is to compare the memory and timing requirements for a vector–matrix product computation using the full vector and the HTD format approaches. Furthermore, we have to evaluate the accuracy of the computation if truncation is performed in the HTD format. Therefore, we consider in the following iteration steps of the Power method. This is not the most efficient solution method for steady-state analysis, but similar iteration steps can be applied in

more advanced iterative techniques and they can be directly used in uniformization for transient analysis. For each model, the solution vector $\boldsymbol{\pi}^{(it)}$ at iteration it is multiplied with

$$\mathbf{P} := \mathbf{I} + \Delta \mathbf{Q}, \quad \text{where } \Delta := 0.999 / \max_{\mathbf{s} \in \mathcal{R}} |\mathbf{Q}(\mathbf{s}, \mathbf{s})|,$$

starting with the uniform distribution in $\boldsymbol{\pi}^{(0)}$, so that we have

$$\boldsymbol{\pi}^{(it)} := \boldsymbol{\pi}^{(it-1)} \mathbf{P} \quad \text{for } it = 1, 2, \dots, \text{maxit}$$

with the associated error vector $\mathbf{r}^{(it)} := \boldsymbol{\pi}^{(it)} - \boldsymbol{\pi}^{(it-1)}$. Note that $\mathbf{r}^{(it)} = \Delta \boldsymbol{\pi}^{(it-1)} \mathbf{Q}$, the scaled residual vector corresponding to the previous iteration vector. Here, `maxit` is the maximum number of iterations and we set `maxit` := 1,000. The numerical experiments are performed on an Intel Core i7 Quad-Core 3.6 GHz processor with 32 GB of main memory.

Table 4.2 contains the results for the availability model. Time is in seconds and Memory indicates the number of allocated real array elements rounded to two decimal digits of accuracy. For the chosen truncation accuracy of $tol \in [1\text{e} - 9, 1\text{e} - 7]$, the norm of the final error vector is the same for the full and HTD representations. Due to the reduced memory requirements of the vector, the compact representation results even in smaller iteration times when d increases. It should be mentioned that the model is not symmetric due to the priority repair strategy but, as it is common in availability models, the probability distribution becomes unbalanced because repair rates are higher than failure rates.

The situation is more ambiguous for the polling example whose results are given in Table 4.3. For the larger configurations, we obtain savings in memory by several orders of magnitude even with the smallest truncation accuracy of tol . Timewise the conventional approach is faster for small configurations, but it is outperformed by the compact representation for larger state spaces (i.e. $d = 6$), if tol is not too small. The largest configuration with $d = 7$ can only be analysed with the compact vector representation.

In Figure 4.3 the ranks of the different matrices forming the HTD are shown for a truncation accuracy of $tol = 1\text{e} - 7$. It can be seen that the ranks remain

Table 4.2: Numerical results for availability models

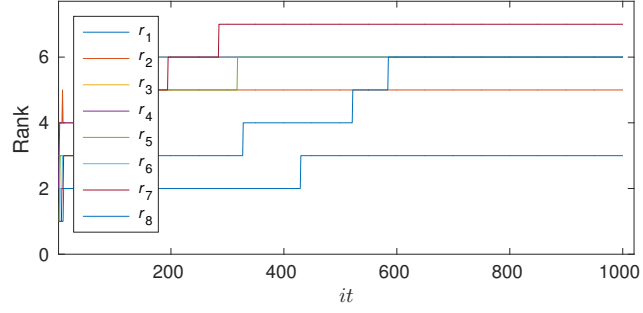
d	Time	Full			tol	Compact		
		Memory	$\ \mathbf{r}^{(\maxit)}\ _2$			Time	Memory	$\ \mathbf{r}^{(\maxit)}\ _2$
3	0	5.4e3	$5e-7$	$1e-7$		1	2.1e3	$7e-7$
				$1e-8$		1	2.3e3	$5e-7$
				$1e-9$		2	3.4e3	$5e-7$
4	0	6.3e4	$3e-6$	$1e-7$		3	2.8e3	$2e-6$
				$1e-8$		4	4.1e3	$3e-6$
				$1e-9$		6	6.9e3	$3e-6$
5	2	7.5e5	$1e-5$	$1e-7$		6	3.7e3	$9e-6$
				$1e-8$		9	5.3e3	$1e-5$
				$1e-9$		13	9.3e3	$1e-5$
6	38	9.0e6	$3e-5$	$1e-7$		13	6.8e3	$3e-5$
				$1e-8$		18	9.4e3	$3e-5$
				$1e-9$		28	1.4e4	$3e-5$
7	513	1.1e8	$7e-5$	$1e-7$		15	6.7e3	$7e-5$
				$1e-8$		28	1.0e4	$7e-5$
				$1e-9$		43	1.6e4	$7e-5$
8	6,329	1.3e9	$2e-6$	$1e-7$		22	9.1e3	$9e-5$
				$1e-8$		37	1.2e4	$9e-6$
				$1e-9$		66	2.6e4	$3e-6$

moderate. The matrices are fairly dense such that a sparse storage of the matrices for the vector representation is not necessary which can be seen in Figure 4.4.

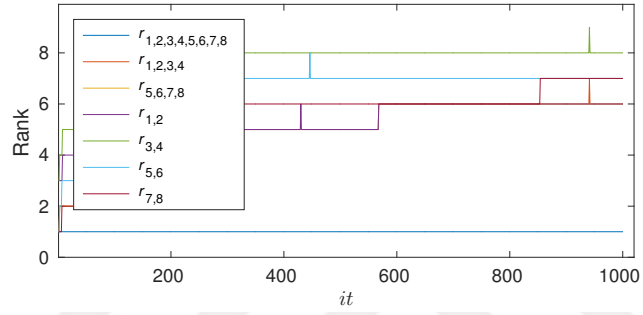
Table 4.3: Numerical results for polling models

d	Time	Full			tol	Compact		
		Memory	$ \mathbf{r}^{(\maxit)} _2$			Time	Memory	$ \mathbf{r}^{(\maxit)} _2$
3	0	8.3e4	4e-6	1e-7		50	4.9e4	4e-6
				1e-8		83	7.2e4	4e-6
				1e-9		108	8.9e4	4e-6
4	5	1.5e6	5e-6	1e-7		285	1.4e5	5e-6
				1e-8		1,175	4.0e5	5e-6
				1e-9		3,272	7.7e5	5e-6
5	103	2.5e7	5e-6	1e-7		409	2.2e5	5e-6
				1e-8		2,522	8.0e5	5e-6
				1e-9		10,951	2.1e6	5e-6
6	1,896	3.8e8	3e-6	1e-7		254	1.8e5	3e-6
				1e-8		2,448	8.8e5	3e-6
				1e-9		19,423	3.6e6	3e-6
7	n/a	5.7e9	n/a	1e-7		196	2.2e5	3e-6
				1e-8		1,831	9.0e5	2e-6
				1e-9		21,668	5.0e6	2e-6

Figure 4.3: Ranks of basis and transfer matrices forming $\pi^{(it)}$, availability $d = 8$

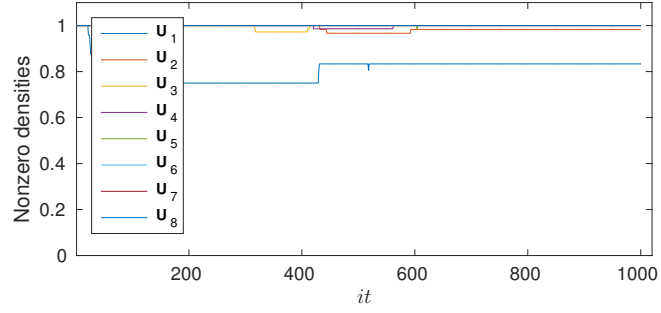


(a) Basis matrices

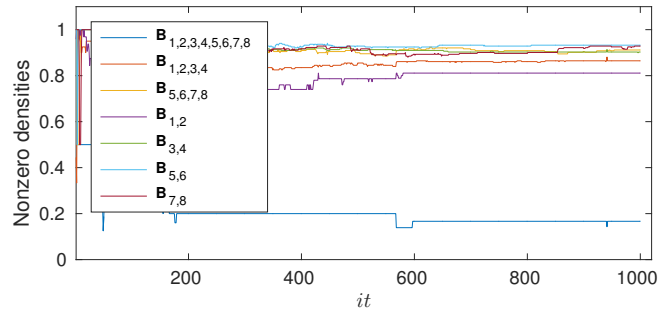


(b) Transfer matrices

Figure 4.4: Densities of basis and transfer matrices forming $\pi^{(it)}$, availability $d = 8$



(a) Basis matrices



(b) Transfer matrices

Chapter 5

Numerical solution of the chemical master equation

A stochastic chemical system with multiple types of molecules interacting through reaction channels can be modelled as a continuous-time Markov chain (CTMC) with a countably infinite multi-dimensional state space. In this context, a state is a row vector with elements representing the number of molecules of each type present in the system. A state transition corresponds to a reaction channel which is expressed by a state change vector and a state-dependent transition rate function. The ODE system describing the time evolution of the probability distribution associated with the CTMC is known as the chemical master equation (CME). The goal of this chapter is to compute the transient probability distribution of the stochastic chemical system at a particular time by solving the CME whose coefficient matrix is the infinitesimal generator of its underlying CTMC. A detailed overview on transient analysis of stochastic chemical systems may be obtained, for instance, from the extensive review by Goutsias and Jenkinson [80].

5.1 Chemical master equation

Analytical solution of the CME is possible only in a few simple cases [70, 91, 93, 152]; therefore, other approaches are considered. A popular approach is to use the stochastic simulation algorithm (SSA) [74, 75] in which sample trajectories are generated by simulating reaction events. SSA is said to be exact in the sense that every reaction event is simulated. Approximation techniques that avoid simulating every reaction event such as tau-leaping methods [76, 133] and system-partitioning methods [34, 79, 132] are suggested to reduce computational complexity by trading-off accuracy. A drawback of simulation techniques is that they require a large number of trajectories to obtain sufficiently accurate estimates and they are inefficient especially for estimating probability distributions.

Another approach is to solve the CME numerically. This is a formidable task since size of the CME increases exponentially with number of dimensions even when an upper bound is applied to the number of molecules in the system. Munsky and Khammash [118] propose the finite state projection (FSP) algorithm to approximate the actual probability distribution. The FSP algorithm gradually expands the truncated state space by adding states that are reachable within a certain number of transitions, and solves the truncated CME until the accuracy of the approximate solution meets a prespecified tolerance. The algorithm uses the number of molecule types, that is, number of dimensions, in limiting reachability during the truncated state space expansion process. Unfortunately, this strategy yields a truncated state space including many states with negligible probability. Furthermore, the matrix scaling and powering method in [144, pp. 416-417] employed to solve the truncated CME becomes inefficient as the size of the system increases. Hence, several variants of the FSP algorithm such as multiple time interval FSP [119], Krylov-FSP [33, 113], optimal FSP [145], and sliding window [150] methods are proposed to solve the CME more efficiently. These methods differ from each other by the choice of the truncated state space and the technique employed to solve the truncated CME. We refer to the survey by Dinh and Sidge [60] for an overview of the FSP algorithm and its variants.

Accurate approximation of the probability distribution requires the truncated state space to be large in many cases. Some numerical methods take a hybrid approach and combine the CME solver with other techniques to improve execution time and memory requirement by trading-off accuracy [45, 90]. Other numerical methods reduce the degrees of freedom, that is, number of unknowns, in the CME by sparse grids [87, 89], discrete orthogonal polynomials [56, 64], and sparse wavelets [92, 95]. In essence, these methods represent the probability distribution as a linear combination of basis functions. Although the degrees of freedom become smaller, these methods still suffer from the curse of dimensionality, that is, memory requirement and computational cost scaling exponentially with number of dimensions.

In order to overcome the curse of dimensionality, low-rank approximation techniques that represent multi-dimensional arrays (or tensors) compactly are employed in many application areas (see, for instance, [82]). Canonical polyadic (CP) decomposition [35, 85] is a compact representation of a tensor, where the multi-dimensional array is given as a sum of Kronecker products of smaller vectors or rank-one tensors. Jahnke and Huisinga [94] project the dynamics of the system onto a manifold of all tensors in CP decomposition format with a predetermined maximal tensor rank. The rank bound is chosen based on the initial distribution and kept constant. However, they advise an adaptive rank control strategy since it is not possible to estimate the rank of a vector in CP decomposition format that approximates the solution of the CME with a desired accuracy. Hegland and Garcke [88] employ the implicit Euler method to solve the CME and store probability vectors in CP decomposition format. Therein, a bound on the ranks is not chosen and they suggest to reduce ranks after algebraic tensor operations using various procedures [102]. However, this approach appears to pose problems in practice since there is no robust and efficient algorithm to truncate CP decomposition to lower rank with any desirable accuracy for tensors with more than two dimensions.

Tucker decomposition [147] represents a d -dimensional tensor as the multiplication of a d -dimensional core tensor and a basis matrix along each dimension. Higher-order singular value decomposition (HOSVD) [109] provides a simple and

nearly optimal solution to approximate a tensor in Tucker decomposition format with a user controlled error. However, memory requirement of the core tensor scales exponentially with number of dimensions. Tensor train (TT) decomposition [123] is a low-rank approximation technique that avoids this exponential dependence on number of dimensions. TT decomposition is extended to quantized TT (QTT) decomposition by introducing additional virtual dimensions in order to decrease memory requirements and time complexity of basic operations. It is possible to represent the solution of the truncated CME compactly in QTT format and reduce memory requirements of solution vectors [61, 62, 97].

Another low-rank approximation technique that admits truncation methods with user controlled approximation error is hierarchical Tucker decomposition (HTD) [81, 84, 104, 105]. HTD may be considered as a generalization of TT decomposition in which basis and core matrices are related through a tree structure with logarithmic depth in the number of dimensions that reduces the complexity of Tucker decomposition by hierarchically splitting the core tensor into core matrices. An advantage of HTD representation is that the multiplication of a compact solution vector in HTD format with a sum of Kronecker products is a well-defined operation [29]. Hence, Kronecker-based Markovian analysis with vectors in HTD format is possible under the separability assumption of transition rates as in [56, 75, 94, 97, 115, 135]. To the best of our knowledge, HTD format has not been employed previously in transient analysis of the CME.

Backward differentiation formulae (BDF) [144, pp. 446-448] are a class of multistep methods to solve ODEs. The k -step BDF (BDFk) method keeps approximations of solutions at k previous time steps and computes the solution at the current time step by solving a linear system. BDFk methods have two major advantages in solving the CME. First, BDFk provides an efficient solver for stiff systems in which the solution contains rapidly decaying transient terms. Stiffness generally manifests itself when reaction rates occur at different time scales, and this is the case for many realistic systems. Second, a prediction of the solution can be given as a linear combination of previous solutions [11, p. 150] and local truncation error can be obtained from the difference between prediction and solution at that step [137]. The prediction vector provides a good estimate of the

solution, and to the contrary of the FSP algorithm, it is possible to choose a truncated state space that includes most of the probability mass without computing the solution of the truncated CME.

We have implemented a BDFk method with adaptive truncation of the state space and vectors in HTD format. BDFk is solved iteratively by Jacobi method which splits the coefficient matrix into diagonal and negated off-diagonal matrices at the outset so that the iteration vector is multiplied by the negated off-diagonal matrix and inverse of the diagonal matrix at each iteration. Computation of the inverse of the diagonal matrix at the outset is convenient. To that end, we obtain the diagonal of the coefficient matrix in HTD format and compute the element-wise reciprocal of this compact vector iteratively using Newton-Schulz method [104]. If an iteration vector is not close to solution, an accurate computation of the vector should not be necessary. Furthermore, the imposed accuracy may bring additional time and memory requirements. Therefore, we propose adaptive truncation strategies in Newton-Schulz and Jacobi iterations that allow relatively inaccurate computation of iteration vectors. Numerical experiments indicate that such adaptive truncation strategies avoid increase in memory requirements and execution time when truncation with fixed error tolerance and fixed rank bound leads to an inefficient CME solver.

The generator matrix underlying the a stochastic chemical system can be described by J transition classes each corresponding to a reaction channel as in the next definition [46].

Definition 5. A pair $(\alpha^{(j)}, \mathbf{v}^{(j)})$ is said to be *transition class* $j \in \{1, \dots, J\}$, where $\alpha^{(j)} : \mathcal{S} \rightarrow \mathbb{R}_{>0}$ and $\mathbf{v}^{(j)} = (v_1^{(j)}, \dots, v_d^{(j)}) \in \mathbb{Z}^{1 \times d}$ are respectively the transition rate function and the state change vector.

We assume the separability of transition rate function α . That is, there exists $\gamma^{(j)} \in \mathbb{R}_{\geq 0}$ and $\alpha_h^{(j)} : \mathcal{S}_h \rightarrow \mathbb{R}_{\geq 0}$ such that

$$\alpha^{(j)}(\mathbf{s}) = \gamma^{(j)} \prod_{h=1}^d \alpha_h^{(j)}(s_h)$$

for $\mathbf{s} = (s_1, \dots, s_d) \in \mathcal{S}$ and $j = 1, \dots, J$. This separability assumption holds for

elementary reactions that are also building blocks in more complicated reaction kinetics [56, 75, 94, 97, 115, 135].

Here, all states in the product state space are reachable. Hence, the generator matrix underlying the CTMC can be represented as a sums of Kronecker products and the generator matrix is a (1×1) block matrix. In this chapter, we omit the block indices and write the index j associated with the transition as a superscript in parentheses above the matrices forming the block as in the previous chapter. Then, the transition matrix $\mathbf{S}_h^{(j)} \in \mathbb{R}_{\geq 0}^{|\mathcal{S}_h| \times |\mathcal{S}_h|}$ is given elementwise as

$$\mathbf{S}_h^{(j)}(s_h, s'_h) = \begin{cases} \alpha_h^{(j)}(s_h) & \text{if } s'_h = s_h + v_h^{(j)} \\ 0 & \text{otherwise} \end{cases}$$

for $s_h, s'_h \in \mathcal{S}_h$, $j \in \{1, \dots, J\}$, and $h \in \{1, \dots, d\}$. Then the infinitesimal generator matrix can be written as

$$\mathbf{Q} = \sum_{j=1}^J \mathbf{S}^{(j)} - \sum_{j=1}^J \mathbf{D}^{(j)}$$

with

$$\mathbf{S}^{(j)} = \gamma^{(j)} \bigotimes_{h=1}^d \mathbf{S}_h^{(j)} \quad \text{and} \quad \mathbf{D}^{(j)} = \gamma^{(j)} \bigotimes_{h=1}^d \text{diag}(\mathbf{S}_h^{(j)} \mathbf{e}) \quad \text{for } j = 1, \dots, J.$$

Having given the infinitesimal generator matrix described by transition classes, now let us turn to the solution of the CME (see (1.1) in Chapter 1) through backward differentiation formulae.

5.2 Backward differentiation formulae

BDF are a class of multistep methods that are widely used to cope with stiffness in the solution of ODEs. In this class of solvers, time is discretized and solutions at intermediate time steps are approximated using backward differences [11, p. 125]. It is not possible to use a BDF solver when the ODE system is countably infinite; however, the ODE system may be truncated by assuming that the probability of

the system being at a state outside the finite truncated state space $\mathcal{R}_n \subset \mathcal{S}$ is zero.

In order to distinct the exact and computed solutions of the CME and avoid using two levels of subscripts when describing the BDF method, we omit t and Δt in the notation, and let vector $\mathbf{p}_{n-i}$ hold the probabilities of the system being at states in $\mathcal{R}_n$ at time $(t_n - i\Delta t)$ for $i = 0, \dots, k$. Then for a constant step size $\Delta t > 0$, the formula of order k BDF (BDFk) at step n is given by

$$\sum_{j=1}^k \frac{1}{j} \nabla^j \mathbf{p}_n = \Delta t \mathbf{p}_n \mathbf{Q}_n \quad \text{with} \quad \mathbf{Q}_n = \mathbf{Q}(\mathcal{R}_n, \mathcal{R}_n)$$

and backward differences are recursively defined as

$$\nabla^0 \mathbf{p}_n = \mathbf{p}_n \quad \text{and} \quad \nabla^{j+1} \mathbf{p}_n = \nabla^j \mathbf{p}_n - \nabla^j \mathbf{p}_{n-1} \quad \text{for } j, n \in \mathbb{Z}_{>0}.$$

Note that BDFk may be rewritten as a linear system in the form

$$\mathbf{p}_n (\mathbf{I} - \theta_k \Delta t \mathbf{Q}_n) = \sum_{j=0}^{k-1} \beta_{k,j} \nabla^j \mathbf{p}_{n-1}, \quad (5.1)$$

where θ_k and $\beta_{k,0}, \dots, \beta_{k,k-1}$ are real numbers defined by BDF order k . For instance, $\theta_5 = 60/137$, $\beta_{5,0} = 1$, $\beta_{5,1} = 77/137$, $\beta_{5,2} = 47/137$, $\beta_{5,3} = 27/137$, and $\beta_{5,4} = 12/137$ when $k = 5$.

The leading term of the BDFk local truncation error can be approximated as

$$\mathbf{r}_n \approx \frac{1}{k+1} (\mathbf{p}_n - \mathbf{p}_n^{(0)}), \quad (5.2)$$

where

$$\mathbf{p}_n^{(0)} = \sum_{j=0}^k \nabla^j \mathbf{p}_{n-1}$$

is the prediction vector [137].

Probability of the system being in a state changes over time. Hence, at each time step, truncated state space $\mathcal{R}_n$ needs to be chosen such that probability of the system being outside $\mathcal{R}_n$ is small. One possible approach is solving (5.1) and

expanding the truncated state space until it includes most of the probability mass as in the FSP algorithm [118]. However, it is possible to take a different and more efficient approach. Prediction vector $\mathbf{p}_n^{(0)}$ is a good estimate of the solution at time $t_n = t_{n-1} + \Delta t$. Hence, truncated state space $\mathcal{R}_n$ captures most of the probability mass at time t_n if $\sum_{\mathbf{s} \notin \mathcal{R}_n} \mathbf{p}_n^{(0)}(\mathbf{s})$ is small. This approach allows to choose the truncated state space before solving (5.1) since the backward difference vectors $\nabla^0 \mathbf{p}_{n-1}, \dots, \nabla^k \mathbf{p}_{n-1}$ are obtained at previous time steps.

Backward differences are also very suitable for allowing to change the step size [137]. Letting $\mathbf{P}$ and $\mathbf{P}^*$ respectively denote matrices with backward differences for old step size Δt and new step size $\Delta t^* \neq \Delta t$ as in

$$\mathbf{P} = [(\nabla^1 \mathbf{p}_{n-1})^T, \dots, (\nabla^k \mathbf{p}_{n-1})^T] \quad \text{and} \quad \mathbf{P}^* = [(\nabla^{*1} \mathbf{p}_{n-1})^T, \dots, (\nabla^{*k} \mathbf{p}_{n-1})^T],$$

new backward differences may be obtained from $\mathbf{P}^* = \mathbf{P} \bar{\mathbf{R}}^* \bar{\mathbf{U}}^*$, where $\bar{\mathbf{R}}^*$ and $\bar{\mathbf{U}}^*$ are $(k \times k)$ matrices given elementwise as

$$\bar{\mathbf{R}}^*(i, i') = \frac{1}{i!} \prod_{j=0}^{i-1} (j - i'(\Delta t^*/\Delta t)) \quad \text{and} \quad \bar{\mathbf{U}}^*(i, i') = \frac{1}{i!} \prod_{j=0}^{i-1} (j - i') \quad \text{for } i, i' = 1, \dots, k.$$

The next section provides a description of the iterative methods with vectors in HTD format that are used in the implementation of BDFk.

5.3 Iterative solution vectors in HTD format

5.3.1 Newton-Schulz method

It is costly to generate each element of a vector in HTD format; hence, an iterative method such as Newton-Schulz [104, p. 28] can be employed to compute the elementwise reciprocal of a vector $\mathbf{x}$ in HTD format as in Algorithm 7. Newton-Schulz can be written as

$$\mathbf{xinv}^{(it)} := \mathbf{xinv}^{(it-1)} + \mathbf{xinv}^{(it-1)} \star \left(\mathbf{1} - \mathbf{x} \star \mathbf{xinv}^{(it-1)} \right) \quad \text{for } it = 1, 2, \dots$$

starting with a vector $\mathbf{xinv}^{(0)}$ with each element having reciprocal of the 2-norm of input vector $\mathbf{x}$. The algorithm stops if relative 2-norm of $(\mathbf{1} - \mathbf{x} \star \mathbf{xinv}^{(it-1)})$ is smaller than tolerance rcp_tol as in `elem_reciprocal.m` of `htucker_toolbox` [104] or if the change in the norm is larger than tolerance rcp_chg_tol and rcp_max_it iterations have already been performed.

Input: Input vector in HTD format: $\mathbf{x}$
Output: Elementwise reciprocal of input vector in HTD format: $\mathbf{xinv}$

```

1: function NEWTON-SCHULZ( $\mathbf{x}$ )
2:   Set each element of  $\mathbf{xinv}$  to  $1/\|\mathbf{x}\|_2$ ;
3:    $it \leftarrow 0$ ;  $nrm\_chg \leftarrow 0$ ;  $rcp\_stop \leftarrow FALSE$ ;
4:   do
5:      $it \leftarrow it + 1$ ;  $\mathbf{xinv}^{(it-1)} \leftarrow \mathbf{xinv}$ ;
6:      $\mathbf{d} \leftarrow \mathbf{1} - \mathbf{x} \star \mathbf{xinv}^{(it-1)}$ ;  $nrm^{(it)} \leftarrow \|\mathbf{d}\|_2/\|\mathbf{1}\|_2$ ;
7:      $\mathbf{xinv} \leftarrow \mathbf{xinv}^{(it-1)} + \mathbf{xinv}^{(it-1)} \star \mathbf{d}$ ;
8:     if  $it > 1$  then
9:        $nrm\_chg \leftarrow nrm^{(it)}/nrm^{(it-1)}$ ;
10:    end if
11:    if  $nrm\_chg > rcp\_chg\_tol$  AND  $it \geq rcp\_max\_it$  then
12:       $rcp\_stop \leftarrow TRUE$ ;
13:    end if
14:    while  $nrm^{(it)} > rcp\_tol$  AND  $rcp\_stop = FALSE$ 
15:      return  $\mathbf{xinv}$ ;
16: end function

```

Algorithm 7: Newton-Schulz method

Note that each iteration of Newton-Schulz requires computing the elementwise multiplication of vectors in HTD format (see Lines 6 and 7 of Algorithm 7). The elementwise multiplication of two vectors in HTD format may be truncated directly while computing basis and transfer matrices as discussed in [104, pp. 24–25]. During truncation, it is possible to impose an accuracy of ϵ on the truncated HOSVD by choosing ranks based on dropping the smallest singular values whose squared sum is less than or equal to $\epsilon^2/(2d - 3)$ as it is done in the addition of two vectors.

5.3.2 Jacobi method

The linear system (5.1) can be solved by the classical point iterative Jacobi method as in Algorithm 8. Jacobi can be written as

$$\mathbf{p}_n^{(it)} := (\mathbf{p}_n^{(rhs)} + \mathbf{p}_n^{(it-1)} \mathbf{N}_n) \star \mathbf{minv}_n \quad \text{for } it = 1, 2, \dots$$

starting with prediction vector $\mathbf{p}_n^{(0)}$, where

$$\mathbf{p}_n^{(rhs)} = \sum_{j=0}^{k-1} \beta_{k,j} \nabla^j \mathbf{p}_{n-1}, \quad \mathbf{N}_n = \theta_k \Delta t \sum_{j=1}^J \mathbf{S}^{(j)}(\mathcal{R}_n, \mathcal{R}_n),$$

and $\mathbf{minv}_n$ is the elementwise reciprocal of

$$\mathbf{m}_n = \mathbf{1} + \theta_k \Delta t \sum_{j=1}^J \mathbf{D}^{(j)}(\mathcal{R}_n, \mathcal{R}_n) \mathbf{e}.$$

At the outset, $\mathbf{minv}_n$ can be computed by Newton-Schulz method (see Line 3 of Algorithm 8). Then iteration vector $\mathbf{p}_n^{(it)}$ and 2-norm of residual $nrm^{(it)}$ are computed at each Jacobi iteration it . Jacobi stops if number of iterations becomes large, iteration vector converges, or stagnation is detected (see Line 12 of Algorithm 8).

Now, we can move to implementation issues regarding HTD-based transient solvers for CTMCs with countably infinite multi-dimensional state spaces.

5.4 Implementation issues

Our implementation [54] extends the one in the previous chapter and relies on basic functions of the `htucker` MATLAB toolbox [104, 105]. It is carried out within the `NSolve` package of APNN Toolbox [1, 14] in C under Linux.

The BDFk method is employed to compute the distribution at time t_f for initial state $\mathbf{s}_0 \in \mathcal{S}$. Time is discretized, and at each time step, truncated state space is chosen, BDFk in (5.1) is solved, and BDFk local truncation error in (5.2)

Input: Prediction vector: $\mathbf{p}_n^{(0)}$, Right-hand-side vector: $\mathbf{p}_n^{(rhs)}$
Truncated state space: $\mathcal{R}_n$, Time step: Δt

Output: Solution of the system: $\mathbf{p}_n$

```

1: function JACOBI( $\mathbf{p}_n^{(0)}$ ,  $\mathbf{p}_n^{(rhs)}$ ,  $\mathcal{R}_n$ ,  $\Delta t$ )
2:    $\mathbf{N}_n \leftarrow \theta_k \Delta t \sum_{j=1}^J \mathbf{S}^{(j)}(\mathcal{R}_n, \mathcal{R}_n)$ ;  $\mathbf{m}_n \leftarrow \mathbf{1} + \theta_k \Delta t \sum_{j=1}^J \mathbf{D}^{(j)}(\mathcal{R}_n, \mathcal{R}_n) \mathbf{e}$ ;
3:    $\mathbf{minv}_n \leftarrow \text{NEWTON-SCHULZ}(\mathbf{m}_n)$ ;
4:    $it \leftarrow 0$ ;  $nrm\_chg \leftarrow 0$ ;  $jac\_stop \leftarrow FALSE$ ;
5:   while  $jac\_stop = FALSE$  do
6:      $it \leftarrow it + 1$ ;  $\mathbf{p}_n^{(it)} \leftarrow (\mathbf{p}_n^{(rhs)} + \mathbf{p}_n^{(it-1)} \mathbf{N}_n) \star \mathbf{minv}_n$ ;
7:      $nrm^{(it)} \leftarrow \|\mathbf{p}_n^{(rhs)} - \mathbf{p}_n^{(it)} (\mathbf{I} - \theta_k \Delta t \mathbf{Q}_n)\|_2$ ;  $\mathbf{p}_n \leftarrow \mathbf{p}_n^{(it)}$ ;
8:     if  $it > 1$  then
9:        $nrm\_chg \leftarrow nrm^{(it)} / nrm^{(it-1)}$ ;
10:    end if
11:    if  $it \geq jac\_max\_it$  OR  $nrm^{(it)} < jac\_tol$  OR  $nrm\_chg > jac\_chg\_tol$  then
12:       $jac\_stop \leftarrow TRUE$ ;
13:    end if
14:  end while
15:  return  $\mathbf{p}_n$ ;
16: end function

```

Algorithm 8: Jacobi method

is computed. If error is not within prespecified error tolerances of err_tol_1 and err_tol_2 , above sequence of operations are repeated for a new step size. Otherwise, backward differences for the next step are computed and the step ends. Following subsections elaborate on details of Algorithm 9.

5.4.1 Choosing truncated state space

Storing solution vectors and matrices compactly, requires the truncated state space to be represented as a union of Cartesian products of subsets of subsystem state spaces [43, 29]. The algorithms proposed for partitioning an arbitrary multi-dimensional state space into Cartesian products in Chapter 2 are relatively time consuming and do not seem to be suitable for our purposes in this context. Efficiently partitioning an arbitrary truncated state space at each time step requires further research. Here, we take a naive approach and let the truncated state space be a Cartesian product of consecutive integers as in sliding windows [150].

Truncated state space $\mathcal{R}_n$ needs to be chosen such that it includes most of the

Input: Truncated state space: $\mathcal{R}_{n-1}$
Backward difference vectors: $\nabla^0 \mathbf{p}_{n-1}, \dots, \nabla^k \mathbf{p}_{n-1}$

Output: Truncated state space: $\mathcal{R}_n$
Backward difference vectors: $\nabla^0 \mathbf{p}_n, \dots, \nabla^k \mathbf{p}_n$

```

1: function BDF_STEP( $\mathcal{R}_{n-1}, \nabla^0 \mathbf{p}_{n-1}, \dots, \nabla^k \mathbf{p}_{n-1}$ )
2:    $accept\_sol \leftarrow FALSE$ 
3:   while  $accept\_sol = FALSE$  do
4:      $\mathbf{p}_n^{(0)} \leftarrow \sum_{j=0}^k \nabla^j \mathbf{p}_{n-1}$ ; Compute truncated state space  $\mathcal{R}_n$  from  $\mathbf{p}_n^{(0)}$ ;
5:     Obtain  $\mathbf{p}_n^{(0)}, \nabla^0 \mathbf{p}_{n-1}, \dots, \nabla^k \mathbf{p}_{n-1}$  for the new truncated state space  $\mathcal{R}_n$ ;
6:      $\mathbf{p}_n^{(rhs)} \leftarrow \sum_{i=0}^{k-1} \beta_{k,i} \nabla^i \mathbf{p}_{n-1}$ ;
7:      $\mathbf{p}_n \leftarrow \text{JACOBI}(\mathbf{p}_n^{(0)}, \mathbf{p}_n^{(rhs)}, \mathcal{R}_n, \Delta t)$ ;
8:      $lte_n \leftarrow \frac{\|\mathbf{p}_n - \mathbf{p}_n^{(0)}\|_2}{k+1}$ ;  $\rho \leftarrow \left( \frac{err\_tol_1 + err\_tol_2}{2 lte_n} \right)^{1/(k+1)}$ ;
9:     if  $err\_tol_1 < lte_n$  AND  $lte_n < err\_tol_2$  then
10:        $accept\_sol \leftarrow TRUE$ ;
11:     else
12:        $\Delta t^* \leftarrow \text{MAX}(\rho \Delta t, \Delta t_{min})$ ;
13:       if  $\Delta t \neq \Delta t^*$  then
14:          $\Delta t \leftarrow \Delta t^*$ ;  $\mathcal{R}_{n-1} \leftarrow \mathcal{R}_n$ ;
15:         Recompute  $\nabla^0 \mathbf{p}_{n-1}, \dots, \nabla^k \mathbf{p}_{n-1}$  for the new time step  $\Delta t$ ;
16:       end if
17:     end if
18:   end while
19:   Compute  $\nabla^0 \mathbf{p}_n, \dots, \nabla^k \mathbf{p}_n$  from  $\nabla^0 \mathbf{p}_{n-1}, \dots, \nabla^k \mathbf{p}_{n-1}$  and  $\mathbf{p}_n$ ;
20:   return ( $\mathcal{R}_n, \nabla^0 \mathbf{p}_n, \dots, \nabla^k \mathbf{p}_n$ );
21: end function

```

Algorithm 9: BDFk method at step n

probability mass at time $(t_n + \Delta t)$. Here, we use the prediction vector to estimate the probability distribution without solving BDFk as discussed in Section 5.2. To that end, for $h' = 1, \dots, d$, we compute the vector $\mathbf{g}_{h'}^{(\mathbf{p}_n^{(0)})} \in \mathbb{R}^{1 \times |\mathcal{R}_{n-1,h}|}$ that is obtained by aggregating all dimensions except dimension h' of vector $\mathbf{p}_n^{(0)}$. The flat representation of the vector $\mathbf{g}_{h'}^{(\mathbf{p}_n^{(0)})}$ can be computed by aggregating the basis matrices of $\mathbf{p}_n^{(0)}$ at dimensions except h' and moving towards the root. Then, assuming that multi-dimensional states are mapped to nonnegative integers lexicographically, we choose $\mathcal{R}_{n,h'}$ such that

$$\mathcal{R}_{n,h'} = \{\min(\tilde{\mathcal{R}}_{n,h'}) - \omega|\tilde{\mathcal{R}}_{n,h'}|, \dots, \max(\tilde{\mathcal{R}}_{n,h'}) + \omega|\tilde{\mathcal{R}}_{n,h'}|\} \cap \mathcal{S}_{h'},$$

where *state_trunc_tol* is a prespecified state truncation tolerance, $\tilde{\mathcal{R}}_{n,h'}$ is a minimal set that satisfies

$$\sum_{s_{h'} \notin \mathcal{R}_{n,h'}} \mathbf{g}_{h'}^{(\mathbf{p}_n^{(0)})}(s_{h'}) < \frac{\text{state_trunc_tol}}{d},$$

and $\omega \in (0, 1)$ is a safety factor. The probability mass that remains outside $\mathcal{R}_n$ due to truncation at each dimension is bounded by *state_trunc_tol*/ d . Hence, probability of the system being outside $\mathcal{R}_n$ is smaller than *state_trunc_tol*, that is, $\mathcal{R}_n$ satisfies

$$\sum_{\mathbf{s} \notin \mathcal{R}_n} \mathbf{p}_n^{(0)}(\mathbf{s}) < \text{state_trunc_tol}.$$

Elements of vectors obtained at the previous step (i.e., backward difference vectors $\nabla^0 \mathbf{p}_{n-1}, \dots, \nabla^k \mathbf{p}_{n-1}$ and prediction vector $\mathbf{p}_n^{(0)}$) correspond to states in the old truncated state space $\mathcal{R}_{n-1}$. These vectors need to be updated so that they become incident to $\mathcal{R}_n$ (see Line 5 of Algorithm 9). The value of an element in the new vector is zero if it corresponds to a state in $\mathcal{R}_n \setminus \mathcal{R}_{n-1}$. Other elements of the new vector have the same values as those of the corresponding elements of the old vector. Rows of new basis and transfer matrices are either zero or can be obtained by copying corresponding rows from old basis and transfer matrices. In conclusion, updating of vectors requires no flops.

5.4.2 Adjusting step size

Local truncation error at each step can be controlled by changing step size. To that end, from (5.2) we compute local truncation error in 2-norm by

$$lte_n = \frac{\|\mathbf{p}_n - \mathbf{p}_n^{(0)}\|_2}{k+1}$$

after Jacobi method provides solution $\mathbf{p}_n$. If lte_n is between error tolerances err_tol_1 and err_tol_2 , then the solution is accepted. Otherwise, candidate step size is chosen as $\Delta t^* = \max(\rho \Delta t, \Delta t_{min})$, where Δt_{min} is minimum step size allowed and

$$\rho = \left(\frac{err_tol_1 + err_tol_2}{2 lte_n} \right)^{1/(k+1)}$$

is optimal step size coefficient for the average of err_tol_1 and err_tol_2 . If candidate step size is different than current step size, step size is updated and backward difference vectors for new step size are computed as discussed in Section 5.2 (see Line 15 of Algorithm 9). We remark that step size is adjusted to compute solution at time t_f when last probability distribution is obtained for time t_n that is sufficiently close to final time t_f .

5.4.3 Truncation strategies

An accuracy of ϵ can be imposed on the truncation of a vector in HTD format by choosing truncation error tolerance as $\epsilon/\sqrt{2d-3}$ [104, 105]. In the original methods of **htucker**, a fixed truncation strategy is considered. Therein, the ranks are determined by a truncation error tolerance and a fixed rank bound. The fixed rank bound is considered to avoid very large ranks that cause the methods to last for an unacceptably long time. The bound needs to be chosen relatively large, e.g., 1,000 in **htucker**, so that the ranks are determined by the truncation error tolerance in most cases.

In fixed truncation strategy, iteration vectors are computed accurately even if they are not close to solution during early iteration steps. However, ranks of

iteration vectors become unnecessarily large in some cases. In order to avoid this, we propose adaptive truncation strategies for Newton-Schulz and Jacobi methods by introducing adaptive rank bounds.

Adaptive truncation strategies need to be devised so that the rank bound is increased only when it is necessary. Based on the observation that the iterative method does not converge or it converges relatively slowly in case of a small rank bound, we increase the bound if the ratio of two consecutive residual norms, i.e., nrm_chg in Algorithms 7 and 8, is larger than a prespecified tolerance. In Newton-Schulz method, rank bound is initialized to one and increased if $nrm_chg > rcp_chg_tol$. In Jacobi method, rank bound is initialized to maximum rank of prediction vector $\mathbf{p}_n^{(0)}$ and increased if $nrm_chg > jac_chg_tol$. In the latter case, the stopping criterion is modified so that Jacobi iterations assume stagnation if $nrm_chg > jac_chg_tol$ at two consecutive iterations.

Truncation of a vector may be inaccurate if rank is determined by a bound instead of a truncation error tolerance. Hence, errors may be underestimated if backward difference vectors, prediction vector, BDFk local truncation error, right-hand side vector of BDFk, and residual vectors are not computed accurately. Therefore, we truncate these vectors only using truncation error tolerances. More precisely, fixed and adaptive rank bounds are only considered while computing elementwise multiplication $\mathbf{x} \star \mathbf{xinv}^{(it-1)}$ (see Line 6 of Algorithm 7), vector $\mathbf{xinv}$ (see Line 7 of Algorithm 7), and iteration vector $\mathbf{p}_n^{(it)}$ (see Line 6 of Algorithm 8).

Regarding error tolerances, the same error tolerance is considered while truncating a vector in both fixed and adaptive truncation strategies. We have respectively chosen $rcp_tol/\sqrt{2d-3}$ and $jac_tol/\sqrt{2d-3}$ as truncation error tolerances in Newton-Schulz and Jacobi methods. Other vectors are truncated with truncation error tolerance $tol/\sqrt{2d-3}$ that imposes an accuracy of tol .

5.5 Numerical results

Numerical experiments are performed on an Intel Core i7 Quad-Core 3.6 GHz processor with 32 GB of main memory; however, only a single processor is used. We consider the BDFk solver of order 5, i.e., $k = 5$, which is also the default order choice in MATLAB's `ode15s` [137]. A time limit of 1,000 seconds is imposed and execution stops at the end of a BDF5 step if execution time has exceeded this limit. Parameters are chosen as either constant or a function of accuracy tolerance tol . Backward difference vectors may include an error of tol ; therefore, we allow the same error in BDF5 and state truncation by letting

$$err_tol_1 = tol/5, \quad err_tol_2 = tol, \quad \text{and} \quad state_tol = tol.$$

Parameters in Newton-Schulz and Jacobi methods are chosen as

$$rcp_tol = jac_tol = tol/2 \quad \text{and} \quad rcp_chg_tol = jac_chg_tol = 1/2$$

so that these methods are efficient yet still relatively accurate. Safety factor ω is set to $1/20$ based on the observation that it is sufficiently small and still yields accurate results. Minimum step size Δt_{min} is set to $1e-6$ based on the assumption that it is sufficiently small.

Numerical experiments are devised to evaluate the effects of using adaptive truncation strategies. We consider three different HTD-based solvers. HTD_F uses fixed truncation strategy in Newton-Schulz and Jacobi methods. HTD_M considers a mixed policy in which adaptive and fixed truncation strategies are used in Newton-Schulz and Jacobi methods, respectively. HTD_A uses adaptive truncation strategies in both iterative methods. In fixed truncation strategies, fixed rank bound is set to 1,000 as in `htucker`.

Besides HTD format, we have implemented the BDF5 method with flat vectors as well as vectors in QTT and transposed QTT (QT3) formats. The flat vector solvers differ from each other by the vector-Kronecker product multiplication algorithms they employ. $FLAT_S$, $FLAT_P$, and $FLAT_{MS}$ respectively employ shuffle algorithm, Pot-RwCl algorithm, and modified shuffle algorithm discussed

in Chapter 3. QTT and QT3 are extensions of TT format, where subsystems of the system are split into virtual subsystems to decrease the memory requirement. A drawback of this quantization is that a vector in QTT or QT3 format can be multiplied by a matrix only in the same format using an iterative method such as the density matrix renormalization group (DMRG) method [124]. DMRG is an optimization-based method that is also used in solving linear systems [97]. At each DMRG iteration (or sweep), core matrices corresponding to consecutive virtual dimensions are aggregated, each aggregated system is solved directly or using an iterative method such as GMRES, and the result of the aggregated system is disaggregated.

Our implementation of the DMRG solver relies on the `dmrg_solve3.m` function of the `tt-toolbox` [3]. We take the same parameters as the default parameters in `dmrg_solve3.m` except for two. Therein, default aggregation order changes in consecutive iterations and aggregated systems are solved directly if system order is smaller than a threshold of 2,500. In our experiments, the former choice leads to slower convergence and the latter choice results in solving relatively large systems directly. Hence, we choose the same aggregation order in all iterations and we solve aggregated systems directly if system order is smaller than 50 [62]. In our experiments, QTT-based solver outperforms QT3-based solver in terms of time, memory, and accuracy. Hence, we do not report results with the latter.

We compare mean number of molecules obtained by CME solvers and SSA in order to evaluate accuracy of the solution. For each model, we have generated 10^8 trajectories using benchmark implementation StochKit [136]. StochKit chooses the optimum SSA method for the given model and reports mean and variance of number of all molecules. Letting $\mu_h^{(SSA)}$ and $\sigma_h^{(SSA)}$ denote mean and standard deviation of number of molecules of type h obtained by simulation, confidence interval [8, pp. 68-71] for a 95% confidence probability is given by

$$(\mu_h^{(SSA)} - \tau_h^{(SSA)}, \mu_h^{(SSA)} + \tau_h^{(SSA)}), \text{ where } \tau_h^{(SSA)} = 1.96 \frac{\sigma_h^{(SSA)}}{10^4}.$$

Then we say that simulation achieves an accuracy of $\max(\tau_h^{(SSA)}/\mu_h^{(SSA)})$. Letting μ_h denote mean number of type h molecules obtained by a CME solver, relative

difference

$$\mu_{dif} = \max_{h=1}^d \left(\frac{\mu_h - \mu_h^{(SSA)}}{\mu_h^{(SSA)}} \right)$$

indicates the error in the computation of mean as long as simulation is more accurate.

We present numerical results in Tables 5.2, 5.4, 5.6, 5.8 and 5.10. First four columns give solver name, accuracy tolerance tol , time t_e at which last solution vector is computed, and number of time steps N_e taken. Column R_{max} lists maximum truncated state space size throughout execution and column r_{max} lists maximum of the ranks of solution vectors. Column ‘It’ reports average number of Jacobi or DMRG iterations per time step depending on the solver, where average is rounded to two decimal digits of accuracy. Columns nnz and ‘Time’ give number of allocated real array elements and execution time in seconds. Next two columns report BDFk and state space truncation errors, where

$$tte = \sum_{n=1}^{N_e} lte_n \quad \text{and} \quad \Delta p = \sum_{n=1}^{N_e} |\mathbf{p}_n \mathbf{e} - \mathbf{p}_{n-1} \mathbf{e}|.$$

Finally, last column reports relative error μ_{dif} if solution at time t_f is computed. Values under R_{max} and nnz columns are rounded to two decimal digits of accuracy, and values under r_{max} , tte , Δp , and μ_{dif} columns are rounded to one decimal digit of accuracy. Values in these columns are reported using scientific notation.

We remark that due to an earlier result, state space truncation error at each step is bounded by total probability mass outside truncated state space if truncated system is solved exactly (see Theorem 2 [118]). However, this value appears to be negative in some models due to numerical errors (see, for instance, [97]). Therefore, instead of computing the probability outside the truncated state space at the end of execution, we report Δp as the state space truncation error. Note that local state space truncation error at step n , i.e., $|\mathbf{p}_n \mathbf{e} - \mathbf{p}_{n-1} \mathbf{e}|$, is dominated by other numerical errors such as BDFk truncation error if probability mass outside truncated state spaces $\mathcal{R}_{n-1}$ and $\mathcal{R}_n$ is large enough. Hence, Δp is in fact an upper bound on the numerical error in 1-norm upon stopping execution of the solver. Throughout the discussion, we refer to Δp as accuracy of the solver.

Numerical results also validate this interpretation since Δp and μ_{dif} are close to each other when simulation is more accurate than the CME solver. We remark that QTT-based solver achieves a higher accuracy than HTD-based solvers for the same accuracy tolerance, and ratio of tte and Δp are quite different for HTD-based solvers and QTT-based solver. This phenomenon seems to be due the structure of QTT and the method chosen to solve BDFk, and it may be the subject of future research. Now, we discuss the models used in experiments and their respective results.

Model 1. (Metabolite synthesis) A biological process associated with metabolite synthesis with two metabolites and two enzymes is considered in [46, 142]. In this model, the molecules interact through nine transition classes given in Table 5.1. Metabolites X_1 and X_2 are respectively synthesized by enzymes X_3 and X_4 through first two transition classes. Third transition class corresponds to consumption of metabolites, and metabolites are degraded through fourth and fifth transition classes. Last four transition classes correspond to production and destruction of enzymes X_3 and X_4 . Here, $d = 4$, $\mathbf{s} = (s_1, s_2, s_3, s_4)$, $J = 9$, and $k_A, k_B, K_I, k_2, \mu, K_R, k_{E_A}, k_{E_B} \in \mathbb{R}_{>0}$.

Table 5.1: Transition classes of metabolite synthesis model

j	$\gamma^{(j)}$	$\alpha_1^{(j)}(s_1)$	$\alpha_2^{(j)}(s_2)$	$\alpha_3^{(j)}(s_3)$	$\alpha_4^{(j)}(s_4)$	$\mathbf{v}^{(j)}$
1	$k_A K_I$	$\frac{1}{s_1 + K_I}$	1	s_3	1	$\mathbf{e}_1^T$
2	$k_B K_I$	1	$\frac{1}{s_2 + K_I}$	1	s_4	$\mathbf{e}_2^T$
3	k_2	s_1	s_2	1	1	$(-\mathbf{e}_1 - \mathbf{e}_2)^T$
4	μ	s_1	1	1	1	$-\mathbf{e}_1^T$
5	μ	1	s_2	1	1	$-\mathbf{e}_2^T$
6	$k_{E_A} K_R$	$\frac{1}{s_1 + K_R}$	1	1	1	$\mathbf{e}_3^T$
7	$k_{E_B} K_R$	1	$\frac{1}{s_2 + K_R}$	1	1	$\mathbf{e}_4^T$
8	μ	1	1	s_3	1	$-\mathbf{e}_3^T$
9	μ	1	1	1	s_4	$-\mathbf{e}_4^T$

In numerical experiments, we let $k_A = k_B = 0.3$, $K_I = 60$, $k_2 = 0.001$, $\mu = 0.002$, $K_R = 30$, and $k_{E_A} = k_{E_B} = 0.02$ as in [142] for initial state $\mathbf{s}_0 = (10, 10, 10, 10)$ and final time $t_f = 10$. Simulation yields the mean with an accuracy of $3e - 5$ in 781 seconds. Table 5.2 presents numerical results for the metabolite synthesis model.

Table 5.2: Numerical results for metabolite synthesis model

Solver	tol	t_e	N_e	R_{max}	r_{max}	It	nnz	Time	tte	Δp	μ_{dif}
HTD _A	1e-6	10.00	203	3.5e5	24	1.6	7.4e4	6	9e-5	2e-3	2e-3
HTD _M		10.00	221	3.5e5	30	1.3	7.6e4	7	1e-4	3e-3	3e-3
HTD _F		6.27	176	2.0e5	1,000	1.3	1.2e8	1,047	8e-5	1e-3	-
QTT		10.00	151	1.3e8	34	2.3	1.4e6	70	5e-5	3e-5	2e-5
FLAT _S		10.00	140	3.5e5	n/a	1.6	3.8e6	3	6e-5	2e-5	2e-5
FLAT _P		10.00	140	3.5e5	n/a	1.6	3.5e6	2	6e-5	2e-5	2e-5
FLAT _{MS}		10.00	140	3.5e5	n/a	1.6	3.8e6	2	6e-5	2e-5	2e-5
HTD _A	1e-7	10.00	225	4.6e5	54	1.8	2.2e5	23	9e-6	1e-4	1e-4
HTD _M		10.00	226	4.6e5	79	1.5	3.5e5	31	9e-6	1e-4	1e-4
HTD _F		2.60	161	1.2e5	1,000	1.3	8.2e7	1,266	7e-6	4e-5	-
QTT		10.00	216	6.7e7	42	2.3	3.2e6	192	8e-6	7e-6	1e-5
FLAT _S		10.00	189	4.7e5	n/a	1.6	5.2e6	6	8e-6	2e-6	2e-5
FLAT _P		10.00	189	4.7e5	n/a	1.6	4.7e6	5	8e-6	2e-6	2e-5
FLAT _{MS}		10.00	189	4.7e5	n/a	1.6	5.2e6	5	8e-6	2e-6	2e-5
HTD _A	1e-8	10.00	317	7.1e5	78	1.8	5.8e5	114	1e-6	2e-5	2e-5
HTD _M		10.00	304	7.1e5	819	1.4	6.5e6	361	1e-6	2e-5	2e-5
HTD _F		1.52	178	1.0e5	1,000	1.5	6.5e7	1,007	8e-7	2e-6	-
QTT		10.00	305	1.3e8	49	2.2	4.5e6	430	1e-6	6e-7	1e-5
FLAT _S		10.00	268	7.1e5	n/a	1.3	7.8e6	10	1e-6	2e-7	2e-5
FLAT _P		10.00	268	7.1e5	n/a	1.3	7.1e6	8	1e-6	2e-7	2e-5
FLAT _{MS}		10.00	268	7.1e5	n/a	1.3	7.8e6	8	1e-6	2e-7	2e-5

HTD_F fails to compute the solution at time t_f since fixed truncation in Newton-Schulz method results in solution vectors with large ranks. Time and memory requirements decrease significantly when adaptive truncation is used in Newton-Schulz instead of the fixed one. When adaptive truncation is also used in Jacobi method, improvements in time and memory become as high as 68% and 91% (for $tol = 1e - 8$), respectively. Figure 5.1 shows that maximum rank of solution vectors changes smoothly over time when adaptive truncation strategies are used.

QTT-based solver is more accurate than HTD-based solvers for the same accuracy tolerance. HTD_A requires around 63% more time and 59% less memory than QTT to achieve the accuracy $3e - 5$ of simulation. Flat solvers are significantly faster and achieve a higher accuracy compared to the solvers with compact vectors. HTD_A is as accurate as FLAT_{MS} for a larger accuracy tolerance (i.e., $tol = 1e - 8$ for HTD_A and $tol = 1e - 6$ for FLAT_{MS}); however, number of nonzeros allocated for FLAT_{MS} is roughly 7 times more than that is allocated for HTD_A.

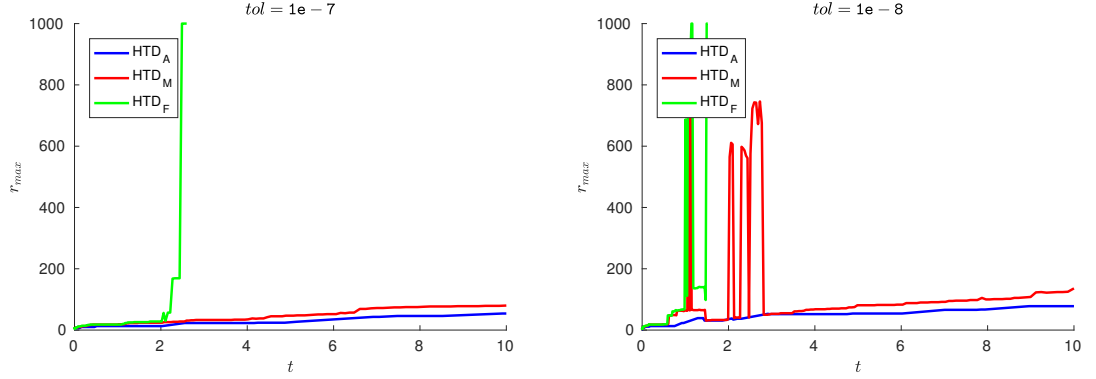


Figure 5.1: Maximum ranks of solution vectors for metabolite synthesis model

Model 2. (Extended toggle switch) A system of stochastic chemical kinetics modelling the biological process associated with another toggle switch is considered in [95]. In this model, X_1 and X_2 are two mutually repressing gene products and they express proteins X_3 and X_4 , respectively. These four molecules interact through eight transition classes given in Table 5.3. First four transition classes respectively correspond to production of X_1 , production of X_2 , destruction of X_1 , and destruction of X_2 . Fifth and sixth transition classes correspond to expression of gene products, and last two transition classes correspond to destruction of gene products. Here, $d = 4$, $\mathbf{s} = (s_1, s_2, s_3, s_4)$, $J = 8$, and $a_{11}, a_{12}, a_{21}, a_{22}, a_3, a_4, a_5, a_6, a_7, a_8 \in \mathbb{R}_{>0}$.

Table 5.3: Transition classes of extended toggle switch model

j	$\gamma^{(j)}$	$\alpha_1^{(j)}(s_1)$	$\alpha_2^{(j)}(s_2)$	$\alpha_3^{(j)}(s_3)$	$\alpha_4^{(j)}(s_4)$	$\mathbf{v}^{(j)}$
1	a_{11}	1	$\frac{1}{s_2^2 + c_{12}}$	1	1	$\mathbf{e}_1^T$
2	a_{12}	$\frac{1}{s_1^2 + c_{22}}$	1	1	1	$\mathbf{e}_2^T$
3	a_3	s_1	1	1	1	$-\mathbf{e}_1^T$
4	a_4	1	s_2	1	1	$-\mathbf{e}_2^T$
5	a_5	s_1	1	1	1	$\mathbf{e}_3^T$
6	a_6	1	s_2	1	1	$\mathbf{e}_4^T$
7	a_7	1	1	s_3	1	$-\mathbf{e}_3^T$
8	a_8	1	1	1	s_4	$-\mathbf{e}_4^T$

In numerical experiments, we let $a_{11} = a_{21} = 10$, $a_{12} = a_{22} = 30$, $a_3 = a_4 = 0.017$, and $a_5 = a_6 = a_7 = a_8 = 0.01$ as in [95] for initial state $\mathbf{s}_0 = (10, 10, 10, 10)$ and final time $t_f = 50$. Simulation yields the mean with an accuracy of $7\mathbf{e} - 5$ in

631 seconds. Table 5.4 presents numerical results for the extended toggle switch model.

Table 5.4: Numerical results for extended toggle switch model

Solver	tol	t_e	N_e	R_{max}	r_{max}	It	nnz	Time	tte	Δp	μ_{dif}
HTD _A	1e-6	50.00	177	7.1e5	26	2.1	1.0e5	5	7e-5	1e-3	2e-3
HTD _M		50.00	177	7.1e5	28	1.7	9.3e4	4	7e-5	1e-3	2e-3
HTD _F		50.00	177	7.1e5	28	1.7	9.3e4	4	7e-5	1e-3	2e-3
QTT		50.00	130	6.7e7	51	2.5	3.7e6	215	4e-5	7e-5	7e-5
FLAT _S		50.00	117	7.5e5	n/a	1.8	8.3e6	5	5e-5	2e-4	6e-5
FLAT _P		50.00	117	7.5e5	n/a	1.8	7.5e6	4	5e-5	2e-4	5e-5
FLAT _{MS}		50.00	117	7.5e5	n/a	1.8	7.5e6	3	5e-5	2e-4	5e-5
HTD _A	1e-7	50.00	162	9.2e5	41	2.1	2.5e5	18	6e-6	2e-4	3e-4
HTD _M		50.00	161	9.2e5	42	1.6	2.5e5	13	6e-6	2e-4	3e-4
HTD _F		50.00	161	9.2e5	42	1.6	2.5e5	13	6e-6	2e-4	3e-4
QTT		50.00	168	2.7e8	56	2.4	5.5e6	409	6e-6	7e-6	6e-5
FLAT _S		50.00	146	9.8e5	n/a	1.6	1.1e7	9	6e-6	2e-6	6e-5
FLAT _P		50.00	146	9.8e5	n/a	1.6	9.8e6	5	6e-6	2e-6	5e-5
FLAT _{MS}		50.00	146	9.8e5	n/a	1.6	9.8e6	5	6e-6	2e-6	5e-5
HTD _A	1e-8	50.00	233	1.2e6	61	2.2	5.5e5	62	9e-7	2e-5	8e-5
HTD _M		50.00	220	1.2e6	65	1.7	6.0e5	48	9e-7	2e-5	8e-5
HTD _F		50.00	223	1.2e6	63	1.8	5.6e5	48	9e-7	2e-5	8e-5
QTT		50.00	214	6.7e7	65	2.2	7.3e6	841	7e-7	8e-7	5e-5
FLAT _S		50.00	194	1.3e6	n/a	1.5	1.4e7	13	8e-7	5e-7	6e-5
FLAT _P		50.00	194	1.3e6	n/a	1.5	1.3e7	8	8e-7	5e-7	5e-5
FLAT _{MS}		50.00	194	1.3e6	n/a	1.5	1.3e7	8	8e-7	5e-7	5e-5

In this model, truncation strategies seem not to be affecting ranks of solution vector. Figure 5.2 shows that maximum rank of solution vectors changes smoothly over time for all HTD-based solvers. Adaptive truncation in Jacobi method increases number of iterations due to additional iterations after rank bounds are increased. However, this increase is not compensated since decrease in ranks of solution vectors is not substantial. Hence, HTD_A is slower than other HTD-based solvers.

QTT-based solver is more accurate than HTD-based solvers for the same accuracy tolerance for this model as well. However, QTT-based solver requires more time and memory to compute the solution as accurately as simulation. Hence, it is outperformed by HTD-based solvers in terms of time, memory, and accuracy. In this model, flat solvers are significantly faster and achieve a higher accuracy compared to the solvers with HTD vectors. HTD_A is as accurate as FLAT_{MS} for a larger accuracy tolerance (i.e., $tol = 1e-7$ for HTD_A and $tol = 1e-6$

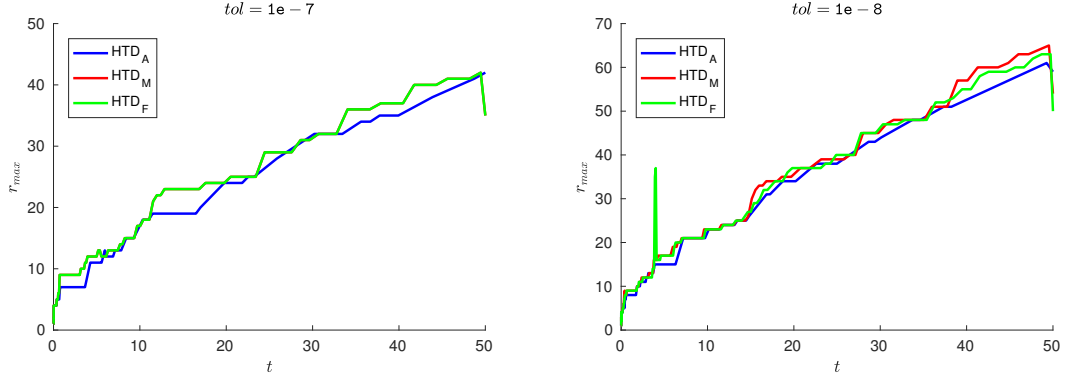


Figure 5.2: Maximum ranks of solution vectors for extended toggle switch model

for FLAT_{MS}); however, number of nonzeros allocated for FLAT_{MS} is roughly 30 times more than that is allocated for HTD_A.

Model 3. (Lambda phage) The biological process associated with life cycle of bacteriophage- λ is a naturally occurring toggle-switch that consists of two phases [87, 94, 138]. Transition from one phase to the other is controlled by competing proteins X_1 and X_2 . In this model, molecules interact through 10 transition classes given in Table 5.5. Proteins X_1 and X_2 are generated through first and third transition classes, respectively. Other proteins are generated through fifth, seventh, and ninth transition classes. Other transition classes correspond to degradation of proteins. Here, $d = 5$, $\mathbf{s} = (s_1, s_2, s_3, s_4, s_5)$, $J = 10$, and $a_1, a_2, a_3, a_4, a_5, \delta_1, \delta_2, \delta_3, \delta_4, \delta_5, b_1, b_2 \in \mathbb{R}_{>0}$.

Table 5.5: Transition classes of lambda phage model

j	$\gamma^{(j)}$	$\alpha_1^{(j)}(s_1)$	$\alpha_2^{(j)}(s_2)$	$\alpha_3^{(j)}(s_3)$	$\alpha_4^{(j)}(s_4)$	$\alpha_5^{(j)}(s_5)$	$\mathbf{v}^{(j)}$
1	$a_1 b_1$	1	$\frac{1}{s_3 + b_1}$	1	1	1	$\mathbf{e}_1^T$
2	δ_1	s_1	1	1	1	1	$-\mathbf{e}_1^T$
3	b_2	$\frac{1}{s_1 + b_2}$	1	1	1	$a_2 s_5 + 1$	$\mathbf{e}_2^T$
4	δ_2	1	s_2	1	1	1	$-\mathbf{e}_2^T$
5	a_3	1	$\frac{s_2}{s_2 + 1}$	1	1	1	$\mathbf{e}_3^T$
6	δ_3	1	1	1	1	1	$-\mathbf{e}_3^T$
7	a_4	1	1	$\frac{s_3}{s_3 + 1}$	1	1	$\mathbf{e}_4^T$
8	δ_4	1	1	1	1	1	$-\mathbf{e}_4^T$
9	a_5	1	1	$\frac{s_3}{s_3 + 1}$	1	1	$\mathbf{e}_5^T$
10	δ_5	1	1	1	$\frac{1}{s_4 + 1}$	s_5	$-\mathbf{e}_5^T$

In numerical experiments, we let $a_1 = 0.5$, $a_2 = 1.0$, $a_3 = 0.15$, $a_4 = 0.3$,

$a_5 = 0.3$, $\delta_1 = 0.0025$, $\delta_2 = 0.0007$, $\delta_3 = 0.0231$, $\delta_4 = \delta_5 = 0.01$, $b_1 = 0.12$, and $b_2 = 0.6$ as in [87] for initial state $\mathbf{s}_0 = (10, 10, 10, 10, 10)$ and final time $t_f = 10$. Simulation yields the mean with an accuracy of $6e - 5$ in 282 seconds. Table 5.6 presents numerical results for the lambda phage model.

Table 5.6: Numerical results for lambda phage model

Solver	tol	t_e	N_e	R_{max}	r_{max}	It	nnz	Time	tte	Δp	μ_{dif}
HTD _A	1e-6	10.00	138	3.8e6	11	1.5	2.8e4	2	5e-5	8e-4	8e-4
HTD _M		10.00	134	3.8e6	28	1.4	6.5e4	4	5e-5	7e-4	7e-4
HTD _F		10.00	139	3.8e6	56	1.4	1.6e5	4	5e-5	7e-4	7e-4
QTT		10.00	109	2.1e9	40	2.5	3.2e6	128	4e-5	5e-5	3e-5
FLAT _S		10.00	105	4.0e6	n/a	1.5	4.8e7	30	4e-5	6e-5	3e-5
FLAT _P		10.00	105	4.0e6	n/a	1.5	4.0e7	16	4e-5	6e-5	2e-5
FLAT _{MS}		10.00	105	4.0e6	n/a	1.5	4.4e7	16	4e-5	6e-5	2e-5
HTD _A	1e-7	10.00	144	7.3e6	30	1.8	1.3e5	9	6e-6	1e-4	1e-4
HTD _M		10.00	148	6.4e6	53	1.3	2.4e5	10	5e-6	9e-5	1e-4
HTD _F		10.00	145	7.3e6	641	1.5	1.4e7	380	6e-6	1e-4	2e-4
QTT		10.00	137	2.1e9	44	2.4	5.0e6	269	5e-6	3e-6	2e-5
FLAT _S		10.00	132	7.3e6	n/a	1.3	8.8e7	63	5e-6	3e-6	3e-5
FLAT _P		10.00	132	7.3e6	n/a	1.3	7.3e7	32	5e-6	3e-6	2e-5
FLAT _{MS}		10.00	132	7.3e6	n/a	1.3	8.1e7	32	5e-6	3e-6	2e-5
HTD _A	1e-8	10.00	194	8.7e6	48	1.8	3.3e5	33	7e-7	2e-5	3e-5
HTD _M		10.00	204	8.7e6	74	1.4	5.0e5	36	8e-7	2e-5	3e-5
HTD _F		7.47	183	5.7e6	824	1.3	1.7e7	1,063	7e-7	1e-5	-
QTT		10.00	181	4.3e9	53	2.2	7.3e6	692	6e-7	5e-7	2e-5
FLAT _S		10.00	174	9.4e6	n/a	1.3	1.1e8	113	7e-7	5e-7	3e-5
FLAT _P		10.00	174	9.4e6	n/a	1.3	9.4e7	58	7e-7	5e-7	2e-5
FLAT _{MS}		10.00	174	9.4e6	n/a	1.3	1.0e8	59	7e-7	5e-7	2e-5

HTD_F fails to compute the solution at time t_f for $tol = 1e - 8$. Time and memory requirements decrease significantly when adaptive truncation is used in Newton-Schulz method. When adaptive truncation is used in Jacobi method, memory requirements decrease around 46% and 34% for accuracy tolerances of $1e - 7$ and $1e - 8$, respectively. It appears that the increase in number of iterations is compensated by the decrease of ranks in solution vectors; hence, execution times with HTD_A and HTD_M are close. Figure 5.3 shows that maximum rank of solution vectors changes smoothly over time only when adaptive truncation is used in Newton-Schulz method.

QTT-based solver requires more time and memory than HTD-based solvers with adaptive truncation in Newton-Schulz to compute the solution as accurately as simulation. Hence, QTT is outperformed by HTD_A and HTD_M in terms of

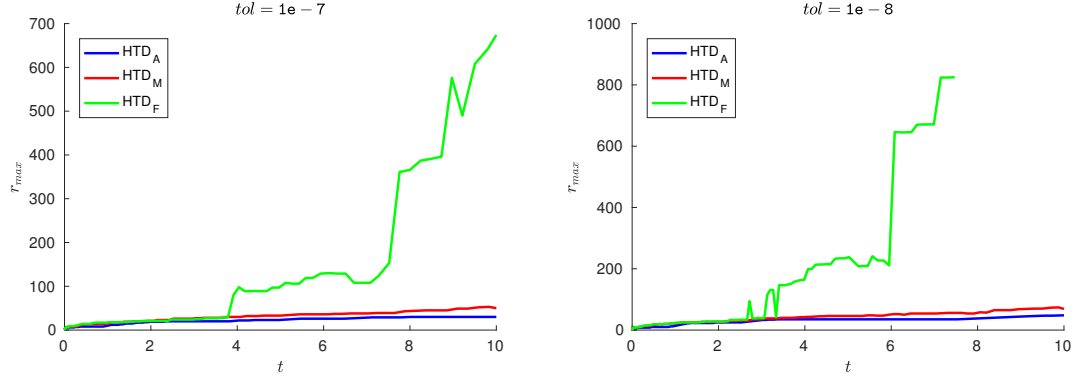


Figure 5.3: Maximum ranks of solution vectors for lambda phage model

time, memory, and accuracy. In this model, for almost the same execution time, FLAT_{MS} computes the solution more accurately compared to HTD_A (i.e., $tol = 1e - 8$ for HTD_A and $tol = 1e - 7$ for FLAT_{MS}); however, number of nonzeros allocated for FLAT_{MS} is roughly 245 times more than that is allocated for HTD_A.

Model 4. (Cascade with five molecules) Consider a system of stochastic chemical kinetics modelling the biological process in which adjacent genes produce protein that enhances expression of succeeding gene [62, 87]. Number of molecules in the cascade model may vary. We consider a cascade model with 5 molecules. Transition classes for 5-dimensional cascade model are given in Table 5.7. Here, $d = 5$, $\mathbf{s} = (s_1, s_2, s_3, s_4, s_5)$, $J = 10$, and $a, b, c, \mu \in \mathbb{R}_{>0}$.

Table 5.7: Transition classes of cascade model five molecules

j	$\gamma^{(j)}$	$\alpha_1^{(j)}(s_1)$	$\alpha_2^{(j)}(s_2)$	$\alpha_3^{(j)}(s_3)$	$\alpha_4^{(j)}(s_4)$	$\alpha_5^{(j)}(s_5)$	$\mathbf{v}^{(j)}$
1	a	1	1	1	1	1	$\mathbf{e}_1^T$
2	μ	s_1	1	1	1	1	$-\mathbf{e}_1^T$
3	b	$\frac{s_1}{bs_1+c}$	1	1	1	1	$\mathbf{e}_2^T$
4	μ	1	s_2	1	1	1	$-\mathbf{e}_2^T$
5	b	1	$\frac{s_2}{bs_2+c}$	1	1	1	$\mathbf{e}_3^T$
6	μ	1	1	s_3	1	1	$-\mathbf{e}_3^T$
7	b	1	1	$\frac{s_3}{bs_3+c}$	1	1	$\mathbf{e}_4^T$
8	μ	1	1	1	s_4	1	$-\mathbf{e}_4^T$
9	b	1	1	1	$\frac{s_4}{bs_4+c}$	1	$\mathbf{e}_5^T$
10	μ	1	1	1	1	s_5	$-\mathbf{e}_5^T$

In the numerical experiments, we let $a = 0.7$, $b = 1.0$, $c = 5.0$, and $\mu = 0.07$ as in [87] for initial state $\mathbf{s}_0 = (10, 10, 10, 10, 10)$ and final time $t_f = 10$. Simulation

yields the mean with an accuracy of $5e - 5$ in 961 seconds. Table 5.8 presents numerical results for the cascade model with five molecules.

Table 5.8: Numerical results for cascade model with five molecules

Solver	tol	t_e	N_e	R_{max}	r_{max}	It	nnz	Time	tte	Δp	μ_{dif}
HTD _A	$1e-7$	10.00	184	2.9e7	17	1.8	5.8e4	5	7e-6	1e-3	1e-3
HTD _M		10.00	187	2.9e7	17	1.6	5.4e4	5	7e-6	1e-3	1e-3
HTD _F		10.00	187	2.9e7	17	1.6	5.6e4	5	7e-6	1e-3	1e-3
QTT		10.00	150	3.4e10	31	2.4	3.1e6	240	5e-6	1e-5	4e-5
FLAT _S		10.00	157	2.9e7	n/a	2.2	3.1e8	731	7e-6	2e-6	3e-5
FLAT _P		10.00	157	2.9e7	n/a	2.2	2.9e8	342	7e-6	2e-6	3e-5
FLAT _{MS}		10.00	157	2.9e7	n/a	2.2	2.9e8	325	7e-6	2e-6	3e-5
HTD _A	$1e-8$	10.00	233	3.5e7	24	1.7	1.4e5	18	9e-7	4e-5	7e-5
HTD _M		10.00	233	3.5e7	26	1.5	1.6e5	14	9e-7	4e-5	7e-5
HTD _F		10.00	231	3.5e7	217	1.6	4.9e6	341	1e-6	4e-5	7e-5
QTT		10.00	209	3.4e10	68	2.7	5.3e6	482	7e-7	2e-6	3e-5
FLAT _S		10.00	214	3.5e7	n/a	1.7	3.8e8	873	8e-7	4e-7	3e-5
FLAT _P		10.00	214	3.5e7	n/a	1.7	3.5e8	427	8e-7	4e-7	3e-5
FLAT _{MS}		10.00	214	3.5e7	n/a	1.7	3.5e8	406	8e-7	4e-7	3e-5
HTD _A	$1e-9$	10.00	292	4.5e7	22	1.8	1.4e5	24	1e-7	8e-6	4e-5
HTD _M		10.00	341	4.5e7	146	1.5	3.7e5	32	1e-7	1e-5	4e-5
HTD _F		9.45	303	4.0e7	437	1.5	1.1e7	1,023	1e-7	7e-6	-
QTT		10.00	290	6.9e10	38	2.1	5.3e6	790	1e-7	9e-8	3e-5
FLAT _S		10.00	274	4.5e7	n/a	1.4	5.0e8	869	1e-7	8e-8	3e-5
FLAT _P		10.00	274	4.5e7	n/a	1.4	4.5e8	452	1e-7	8e-8	3e-5
FLAT _{MS}		10.00	274	4.5e7	n/a	1.4	4.5e8	431	1e-7	8e-8	3e-5

HTD_F fails to compute the solution at time t_f for accuracy tolerance of $1e - 9$. Adaptive truncation in Newton-Schulz method decreases time and memory requirements significantly. Adaptive truncation in Jacobi method decreases maximum of ranks of solution vectors as well as time and memory requirements for $tol = 1e - 9$. However, it has a relatively limited effect for accuracy tolerances of $tol = 1e - 7$ and $tol = 1e - 8$. Figure 5.4 shows that maximum rank of solution vectors changes smoothly over time only when adaptive truncation is used in Newton-Schulz method.

QTT-based solver requires more time and memory than HTD-based solvers with adaptive truncation in Newton-Schulz method to compute the solution as accurately as simulation. In this model, flat solvers achieve higher accuracy; however, the time and memory requirements are significantly larger than that of the HTD-based solvers.

Model 5. (Cascade with six molecules) Consider a system of stochastic chemical

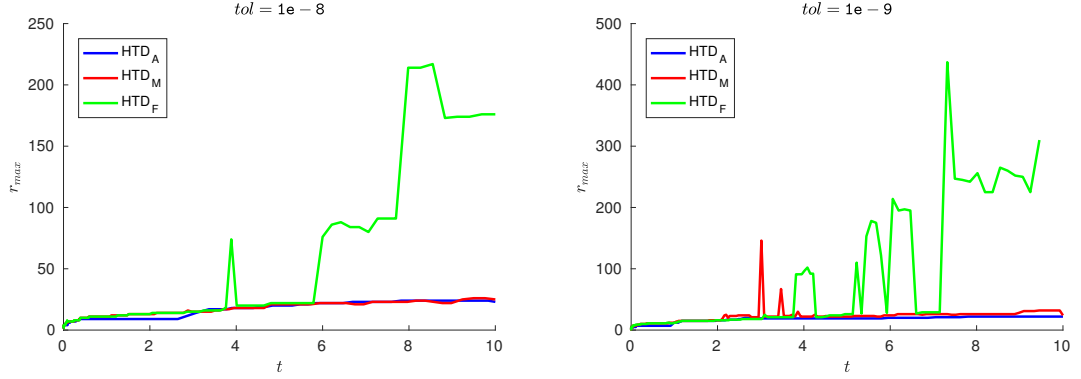


Figure 5.4: Maximum ranks of solution vectors for cascade model with five proteins

kinetics modelling the biological process in which adjacent genes produce protein that enhances expression of succeeding gene [62, 87]. Number of molecules in the cascade model may vary. We consider a cascade model with six molecules. Transition classes for 6-dimensional cascade model are given in Table 5.9. Here, $d = 6$, $\mathbf{s} = (s_1, s_2, s_3, s_4, s_5, s_6)$, $J = 12$, and $a, b, c, \mu \in \mathbb{R}_{>0}$.

Table 5.9: Transition classes of cascade model with six molecules

j	$\gamma^{(j)}$	$\alpha_1^{(j)}(s_1)$	$\alpha_2^{(j)}(s_2)$	$\alpha_3^{(j)}(s_3)$	$\alpha_4^{(j)}(s_4)$	$\alpha_5^{(j)}(s_5)$	$\alpha_6^{(j)}(s_6)$	$\mathbf{v}^{(j)}$
1	a	1	1	1	1	1	1	$\mathbf{e}_1^T$
2	μ	s_1	1	1	1	1	1	$-\mathbf{e}_1^T$
3	b	$\frac{s_1}{bs_1+c}$	1	1	1	1	1	$\mathbf{e}_2^T$
4	μ	1	s_2	1	1	1	1	$-\mathbf{e}_2^T$
5	b	1	$\frac{s_2}{bs_2+c}$	1	1	1	1	$\mathbf{e}_3^T$
6	μ	1	1	s_3	1	1	1	$-\mathbf{e}_3^T$
7	b	1	1	$\frac{s_3}{bs_3+c}$	1	1	1	$\mathbf{e}_4^T$
8	μ	1	1	1	s_4	1	1	$-\mathbf{e}_4^T$
9	b	1	1	1	$\frac{s_4}{bs_4+c}$	1	1	$\mathbf{e}_5^T$
10	μ	1	1	1	1	s_5	1	$-\mathbf{e}_5^T$
11	b	1	1	1	1	$\frac{s_5}{bs_5+c}$	1	$\mathbf{e}_6^T$
12	μ	1	1	1	1	1	s_6	$-\mathbf{e}_6^T$

In the numerical experiments, we let $a = 0.7$, $b = 1.0$, $c = 5.0$, and $\mu = 0.07$ as in [87] for initial state $\mathbf{s}_0 = (10, 10, 10, 10, 10, 10)$ and final time $t_f = 10$. Simulation yields the mean with an accuracy of $5e - 5$ in 1,145 seconds. Table 5.10 presents numerical results for the cascade model.

Table 5.10: Numerical results for cascade model with six molecules

Solver	tol	t_e	N_e	R_{max}	r_{max}	It	nnz	Time	tte	Δp	μ_{dif}
HTD _A	1e-7	10.00	171	8.9e8	24	2.0	8.6e4	11	6e-6	2e-3	2e-3
HTD _M		10.00	180	8.9e8	26	1.7	8.8e4	11	7e-6	2e-3	2e-3
HTD _F		10.00	180	8.9e8	224	1.7	2.4e6	135	7e-6	2e-3	2e-3
QTT		10.00	151	1.1e12	31	2.4	4.9e6	409	5e-6	1e-5	6e-5
FLAT _S		2.17	108	1.9e8	n/a	1.4	2.1e9	1,174	4e-6	2e-6	-
FLAT _P		2.68	113	2.4e8	n/a	1.4	2.4e9	1,030	5e-6	2e-6	-
FLAT _{MS}		2.81	114	2.4e8	n/a	1.4	2.4e9	1,018	5e-6	2e-6	-
HTD _A	1e-8	10.00	222	1.1e9	42	1.8	2.6e5	31	8e-7	1e-4	1e-4
HTD _M		10.00	227	1.1e9	41	1.6	2.0e5	33	9e-7	1e-4	2e-4
HTD _F		7.06	210	8.9e8	487	1.5	1.2e7	1,076	8e-7	7e-5	9e-3
QTT		10.00	213	4.4e12	37	2.3	7.3e6	785	8e-7	1e-6	5e-5
FLAT _S		1.31	127	1.1e8	n/a	1.3	1.2e9	1,043	5e-7	2e-7	-
FLAT _P		1.80	139	2.0e8	n/a	1.3	2.0e9	1,013	5e-7	2e-7	-
FLAT _{MS}		1.92	141	2.4e8	n/a	1.3	2.4e9	1,077	5e-7	2e-7	-
HTD _A	1e-9	10.00	297	1.5e9	53	1.6	5.4e5	87	1e-7	3e-5	8e-5
HTD _M		10.00	294	1.5e9	64	1.4	6.5e5	83	1e-7	3e-5	8e-5
HTD _F		6.59	292	1.1e9	259	1.3	3.9e7	2,696	1e-7	2e-5	-
QTT		6.57	276	8.8e12	41	2.2	8.4e6	1,017	1e-7	1e-7	-
FLAT _S		0.79	152	6.7e7	n/a	1.3	7.4e8	1,016	6e-8	3e-8	-
FLAT _P		1.15	175	1.9e8	n/a	1.2	1.9e9	1,008	7e-8	3e-8	-
FLAT _{MS}		1.19	177	1.9e8	n/a	1.2	1.9e9	1,008	7e-8	3e-8	-

HTD_F fails to compute the solution at time t_f for accuracy tolerances of $1e-8$ and $1e-9$. Adaptive truncation in Newton-Schulz method decreases time and memory requirements significantly. Adaptive truncation in Jacobi method slightly decreases maximum of ranks of solution vectors only for $tol = 1e-9$; however, it has a relatively limited effect on execution time and memory requirement. Figure 5.5 shows that maximum rank of solution vectors changes smoothly over time only when adaptive truncation is used in Newton-Schulz method.

QTT-based solver requires more time and memory than HTD-based solvers with adaptive truncation in Newton-Schulz method to compute the solution as accurately as simulation; however, QTT-based solver is slightly more accurate. In this model, flat solvers fail to compute the solution at the final time instant and number of nonzeros allocated are immense compared to that of solvers with compact vectors.

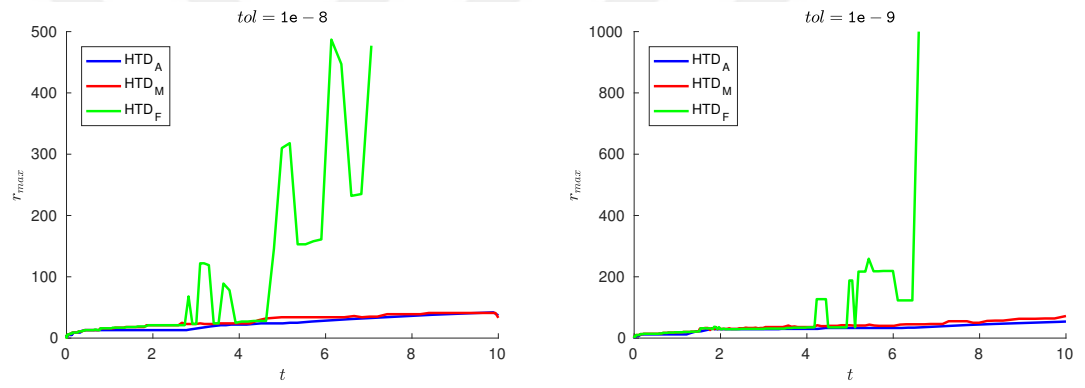


Figure 5.5: Maximum ranks of solution vectors for cascade model with six molecules

Chapter 6

A multi-class MAP/PH/ c queue with acyclic PH retrials

Retrial queues are queueing systems in which an arriving customer who finds all servers busy joins an infinite retrial queue (called the orbit) and retries to receive service. Such systems have been considered in various application areas such as call centers, computer networks, and telecommunication systems. Detailed overviews and bibliographical information about retrial queues may be obtained from the surveys by Gómez-Corral [78] and by Kim and Kim [101], the books by Artalejo and Gómez-Corral [9] and by Falin and Templeton [66], and also from the bibliographies by Artalejo [6, 7].

Criteria based on Lyapunov functions have been widely used for the stability analysis of retrial queues. The following is a review to that end in which the list of references is not exhaustive but covers many related papers. For single- and multi-class M/M/ c queues with exponential retrials respectively in [66] and [141], simple linear Lyapunov functions are used to show the sufficiency and necessity of an ergodicity condition. Artalejo and Phung-Duc [10] extend a single-class M/M/ c queue with exponential retrial by allowing outgoing calls. Sakurai and Phung-Duc [134] consider a more general model than that in [10] with multiple types of outgoing calls. In both systems, simple linear Lyapunov functions are

used in the sufficiency proofs of ergodicity conditions.

Analysis of a system in which arrival or service processes include phases is more complicated. For such systems, one possible approach is to obtain some of the parameters of the Lyapunov function using algebraic properties of the blocks of an auxiliary M/G/1 type matrix (see [121]). Phung-Duc and Kawanishi [126] consider an M/M/ c retrial queue with exponential retrials and after-call work, and allow for customer abandonment in [127]. Diamond and Alfa [57] consider a retrial queue in which customers arrive according to a Markovian arrival processes (MAP) [120], service times follow phase-type (PH) [121] and retrial times follow exponential distributions. For these three systems, the auxiliary M/G/1 type matrices are obtained from their classical queueing system counterparts in which customers join infinite waiting lines instead of orbits. Dudin and Klimenok [63] define asymptotically quasi-Toeplitz Markov chains (AQTMCs) whose blocks converge to the blocks of an M/G/1 type matrix as the level numbers increase. The sufficiency of an ergodicity condition is then obtained by choosing a Lyapunov function whose parameters are determined using the results for the blocks in the M/G/1 type matrix. They also show that a single-server queue with batch MAP (BMAP) arrivals, semi-Markovian service process, and exponential retrials is in the class of AQTMCs. Breur et al. [17] show that a BMAP/PH/ c queue with exponential retrials is in the class of AQTMCs and give a sufficient condition for its ergodicity. He et al. [86] consider a BMAP/PH/ c retrial queue with a waiting line and PH retrials. They obtain a sufficient ergodicity condition by choosing a Lyapunov function whose parameters follow from a classical BMAP/PH/ c queue with no retrials.

Artalejo and Gómez-Corral [9, p. 33] indicate that mathematical analysis of multi-class retrial queues in which all customers can join an orbit is more difficult compared to its single-class counterpart since its joint queue length process is a random walk on the multi-dimensional integer lattice. Expected waiting time expressions are given for two- and multi-class M/G/1 queues with batch arrivals and exponential retrials in [106] and [65], respectively. Avrachenkov et al. [13] consider an M/G/ c queue with waiting lines and constant retrial policy in which only one customer in the orbit can attempt to get service. Shin and

Moon [141] show that the stationary distribution of a multi-class $M/M/c$ queue with exponential retrials converges to that of a classical multi-class $M/M/c$ queue with discriminatory random order service policy as retrial rates tend to infinity, and they present approximation formulae for some performance measures. Kim [99] considers a multi-server multi-class retrial queue in which customers arrive according to a class-dependent Poisson process, service and retrial times follow exponential distributions, and each server can serve a specific class of customers. They obtain a necessary and sufficient condition for positive recurrence by using the fluid limit approach. There are explicit results for a multi-class retrial queue with multiple servers and a few papers take a computational approach. The retrial queue in [39] has two customer classes in which customers either join an infinite waiting line or a finite orbit depending on their class if all servers are busy upon arrival. Therein, this system is modeled as an LDQBD and solved using the matrix-analytic method in [122]. Choi et al. [40] obtain several performance measures; they consider a different system than the one in [39] in that the orbit is infinite and the waiting line is finite. Gharbi et al. [73] model a finite source retrial system using generalized stochastic Petri nets and analyze it with the embedded Markov chain resolution algorithm in [38].

The analysis of a queueing system with PH retrials requires a state representation keeping the number of customers in all retrial phases. This leads to a random walk on the multi-dimensional integer lattice with more than one infinite dimension as for multiple customer classes. Therefore, there are only a handful of papers that consider PH retrials. In those, either simple arrival and service processes are assumed, or a methodology for numerical analysis is not proposed. $M/M/c$ queues with PH retrials appear in [107, 139, 140]. Kumar et al. [107] analyze the waiting time distribution, and Shin and Moon [140] present approximation formulae for the distributions of numbers of customers in service and orbit. Shin [139] describes the model with two retrial phases as an LDQBD and gives an algorithmic solution. In [58], a stability condition for an $M/PH/1$ queue with PH retrials is obtained. Besides, a method is proposed for approximating its stationary distribution and waiting time moments. In [8], a $MAP/PH/c$ queue with PH retrials is numerically analyzed with a finite source assumption. Besides

giving a sufficient condition for ergodicity, He et al. [86] show that the condition is also necessary for the ergodicity of a BMAP/PH/ c queue with a waiting line and PH retrials. However, a methodology for numerical analysis is not proposed. Finally, Chakravarthy [36] studies a MAP/PH/ c queue with PH retrials via simulation due to its complexity. To the best of our knowledge, a multi-class queueing system with MAP arrivals, PH services, and PH retrials has not been analyzed previously.

In this chapter, a multi-class MAP/PH/ c retrial queueing system with acyclic PH retrials is considered. The acyclic PH distribution is a subclass of the PH distribution, but is considered to be as powerful as a general PH distribution, since both are dense in the set of nonnegative distributions (see, for instance, [32]). Therefore the system under consideration is quite general. A necessary and sufficient condition for its ergodicity is obtained from criteria based on drifts by choosing appropriate Lyapunov functions. The system includes multiple customer classes and construction of a useful auxiliary M/G/1 type matrix is not obvious. Hence, it seems that the approach taken to choose the Lyapunov function for the BMAP/PH/ c retrial queue in [86] is not applicable when there are multiple customer classes. The countably infinite state space of the model is truncated with the help of the Lyapunov function chosen in the sufficiency proof so that a finite state space including at least a given steady-state probability mass is determined [44]. Then the truncated model is described as a multi-dimensional Markov chain (MC) and a Kronecker representation of its underlying infinitesimal generator matrix is formed in a similar manner to those models in [30, 46]. Finally, the steady-state distribution of the truncated model is computed iteratively using successive over-relaxation (SOR) [149]. Here, the truncated model is not modeled as an infinite LDQBD and is not solved using the matrix-analytic method of Bright and Taylor [18] although it is possible to do so by choosing an appropriate level definition as in [30, 46, 55]. The reason for this choice is that the method does not scale well as the number of dimensions in the multi-dimensional MC increases. This is due to the increase in the order of the diagonal blocks as the level number increases in multi-dimensional MCs.

6.1 Mathematical model

The system under consideration is a multi-class MAP/PH/ c queue with acyclic PH retrials, where c is the number of servers. We model this system as a multi-dimensional MC and give the generator matrix of the underlying Markov process.

Recall that a MAP can be viewed as a counting process or an irreducible MC with some marked transitions (describing arrivals) as in [32]. We will be using the definition of a MAP with the latter interpretation given below.

Definition 6. MAP with representation $(\mathbf{C}, \mathbf{D})$ of order m is an irreducible MC with a state space of size m and irreducible generator matrix $(\mathbf{C} + \mathbf{D})$, where $\mathbf{C}$ is a nonsingular matrix with negative diagonal and nonnegative off-diagonal elements and $\mathbf{D} \geq 0$.

A state of the MAP in Definition 6 is said to be a phase. Without loss of generality, we assume that the phases of the MAP are numbered 1 through m . The MAP characterizes a stochastic process, where $\mathbf{C}$ includes transitions without an arrival and $\mathbf{D}$ includes transitions with one customer arrival. The definition of PH distribution which will be used to model service and retrial times is given next.

Definition 7. Let $m \in \mathbb{Z}_{\geq 0}$, $\boldsymbol{\beta} \in \mathbb{R}_{\geq 0}^{1 \times m}$, and $\mathbf{T} \in \mathbb{R}^{m \times m}$ be a nonsingular matrix with negative diagonal and nonnegative off-diagonal elements. A PH distribution with representation $(\boldsymbol{\beta}, \mathbf{T})$ of order m is the distribution of time until absorption in a finite state space MC with generator matrix

$$\hat{\mathbf{T}} = \begin{bmatrix} \mathbf{T} & \mathbf{T}^0 \\ \mathbf{0} & 0 \end{bmatrix}_{(m+1) \times (m+1)}$$

and initial probability vector $(\boldsymbol{\beta}, 1 - \boldsymbol{\beta}\mathbf{e}) \in \mathbb{R}_{\geq 0}^{1 \times (m+1)}$.

Without loss of generality, we assume that the state space of $\hat{\mathbf{T}}$ is $\{1, \dots, m+1\}$, where $m+1$ is the absorbing state and the other states are transient. The transient states in a PH distribution are called phases. We assume that the process does

not start in the absorbing state; hence, $\beta e = 1$ holds. Since $\hat{T}$ is the generator matrix of a MC, $T^0 = -Te$ is a nonnegative column vector. Next, the definition of acyclic PH distribution is given.

Definition 8. PH distribution with representation (β, T) is said to be acyclic if its states can be ordered in such a way that T is an upper-triangular matrix.

We consider a retrial queueing system with $c \geq 1$ homogeneous servers and $K \geq 1$ customer classes. Customers of class $k \in \{1, \dots, K\}$ arrive according to MAP with representation (C_k^A, D_k^A) of order m_k^A . Since $(C_k^A + D_k^A)$ is irreducible by Definition 6, there exists a nonnegative row vector $\theta_k^A \in \mathbb{R}_{\geq 0}^{1 \times m_k^A}$ such that $\theta_k^A(C_k^A + D_k^A) = 0$ and $\theta_k^A e = 1$. Furthermore, the average arrival rate is given by $\lambda_k = \theta_k^A D_k^A e$. If all servers are busy upon arrival, an arriving customer of class k joins orbit k and retries to capture a server after a random amount of time. A retrial customer is blocked if it attempts to receive service when there are no idle servers. The service time of a class k customer follows PH distribution with representation (β_k^S, T_k^S) of order m_k^S and $T_k^{S,0} = -T_k^S e$. The retrial time of a class k customer follows acyclic PH distribution with representation (β_k^R, T_k^R) of order m_k^R and $T_k^{R,0} = -T_k^R e$. Hence, we assume that T_k^R is upper-triangular. For a customer of class k , the average service rate is given by $\mu_k = [-\beta_k^S (T_k^S)^{-1} e]^{-1}$ and the average retrial rate is given by $\delta_k = [-\beta_k^R (T_k^R)^{-1} e]^{-1}$.

In [36], a single-class MAP/PH/ c queue with PH retrials is modeled using a multi-dimensional MC. The multi-class counterpart can also be modeled similarly. To that end, we let $X_k(t)$, $X_{b_k^R + i_k^R}(t)$, and $X_{b_k^S + i_k^S}(t)$ respectively denote the phase of the arrival process of class k customers, the number of class k retrial customers in phase i_k^R , and the number of busy servers serving class k customers in phase i_k^S for $i_k^R = 1, \dots, m_k^R$ and $i_k^S = 1, \dots, m_k^S$, where

$$m^R = \sum_{k=1}^K m_k^R, \quad m^S = \sum_{k=1}^K m_k^S, \quad b_k^R = K + \sum_{k'=1}^{k-1} m_{k'}^R, \quad b_k^S = K + m^R + \sum_{k'=1}^{k-1} m_{k'}^S.$$

Then the multi-dimensional MC $X(t) = \{X_1(t), \dots, X_{K+m^R+m^S}(t) : t \geq 0\}$ has the state space $\mathcal{S} = \mathcal{S}^A \times \mathcal{S}^R \times \mathcal{S}^S$, where $\mathcal{S}^A = \times_{k=1}^K \{1, \dots, m_k^A\}$, $\mathcal{S}^R = \mathbb{Z}_{\geq 0}^{m^R}$, $\mathcal{S}^S = \{\mathbf{y} = (y_1, \dots, y_{m^S}) \in \mathbb{Z}_{\geq 0}^{m^S} \mid \mathbf{y}e \leq c\}$, and a possible state representation of

the model is $\mathbf{x} = (x_1, \dots, x_{K+m^R+m^S}) \in \mathcal{S}$. We let $\mathbf{n}(\mathbf{x})$ denote the number of busy servers in state $\mathbf{x}$; that is, $\mathbf{n}(\mathbf{x}) = \sum_{k=1}^K \sum_{i=1}^{m_k^S} x_{b_k^S+i}$. Set $\mathcal{S}^S$ is defined so that the number of busy servers does not exceed the number of servers, and its size is given by

$$|\mathcal{S}^S| = \sum_{i=0}^c \frac{(i + m^S - 1)!}{i! (m^S - 1)!}.$$

Note that another possible approach for modeling this system is to keep the phase of each server in a single but different dimension of the state. However, that approach leads to a larger state space as discussed in [100, 131].

Now, we give the generator matrix underlying the model in terms of matrices associated with arrival, service, and retrial. Matrix $\mathbf{Q}_k^A$ includes transitions associated with arrival of class k customers and is given elementwise as

$$\mathbf{Q}_k^A(\mathbf{x}, \mathbf{y}) = \begin{cases} \mathbf{C}_k^A(x_k, i) & \text{if } i \neq x_k \text{ and } \mathbf{y} = \mathbf{x} + (i - x_k)\mathbf{e}_k^T \\ \mathbf{D}_k^A(x_k, i)\boldsymbol{\beta}_k^R(j) & \text{if } \mathbf{n}(\mathbf{x}) = c \text{ and } \mathbf{y} = \mathbf{x} + (i - x_k)\mathbf{e}_k^T + \mathbf{e}_{b_k^R+j}^T \\ \mathbf{D}_k^A(x_k, i)\boldsymbol{\beta}_k^S(j') & \text{if } \mathbf{n}(\mathbf{x}) < c \text{ and } \mathbf{y} = \mathbf{x} + (i - x_k)\mathbf{e}_k^T + \mathbf{e}_{b_k^S+j'}^T \\ 0 & \text{otherwise} \end{cases}$$

for $i = 1, \dots, m_k^A$, $j = 1, \dots, m_k^R$, $j' = 1, \dots, m_k^S$, and $\mathbf{x}, \mathbf{y} \in \mathcal{S}$, where $\mathbf{e}_k$ represents the k th principal axis vector. Matrix $\mathbf{Q}_k^R$ includes transitions associated with retrial of class k customers and is given elementwise as

$$\mathbf{Q}_k^R(\mathbf{x}, \mathbf{y}) = \begin{cases} x_{b_k^R+i}\mathbf{T}_k^R(i, j) & \text{if } i \neq j \text{ and } \mathbf{y} = \mathbf{x} - \mathbf{e}_{b_k^R+i}^T + \mathbf{e}_{b_k^R+j}^T \\ x_{b_k^R+i}\mathbf{T}_k^{R,0}(i)\boldsymbol{\beta}_k^S(j') & \text{if } \mathbf{n}(\mathbf{x}) < c \text{ and } \mathbf{y} = \mathbf{x} - \mathbf{e}_{b_k^R+i}^T + \mathbf{e}_{b_k^S+j'}^T \\ 0 & \text{otherwise} \end{cases}$$

for $i, j = 1, \dots, m_k^R$, $j' = 1, \dots, m_k^S$, and $\mathbf{x}, \mathbf{y} \in \mathcal{S}$. Finally, matrix $\mathbf{Q}_k^S$ includes transitions associated with service of class k customers and is given elementwise as

$$\mathbf{Q}_k^S(\mathbf{x}, \mathbf{y}) = \begin{cases} x_{b_k^S+i}\mathbf{T}_k^S(i, j) & \text{if } i \neq j \text{ and } \mathbf{y} = \mathbf{x} - \mathbf{e}_{b_k^S+i}^T + \mathbf{e}_{b_k^S+j}^T \\ x_{b_k^S+i}\mathbf{T}_k^{S,0}(i) & \text{if } \mathbf{y} = \mathbf{x} - \mathbf{e}_{b_k^S+i}^T \\ 0 & \text{otherwise} \end{cases}$$

for $i, j = 1, \dots, m_k^S$ and $\mathbf{x}, \mathbf{y} \in \mathcal{S}$. Then the generator matrix underlying $X(t)$ becomes

$$\mathbf{Q} = \mathbf{Q}^{off} + \text{diag}(-\mathbf{Q}^{off}\mathbf{e}) \quad \text{with} \quad \mathbf{Q}^{off} = \sum_{k=1}^K (\mathbf{Q}_k^A + \mathbf{Q}_k^R + \mathbf{Q}_k^S).$$

Example 4. Now consider an example with $K = 2$ and $c = 2$. Let the vectors and matrices describing the arrivals, services, and retrials be given by

$$\begin{aligned} \mathbf{C}_1^A &= \begin{bmatrix} -0.8 & 0.8 \\ 0 & -0.8 \end{bmatrix}_{2 \times 2}, \mathbf{D}_1^A = \begin{bmatrix} 0 & 0 \\ 0.8 & 0 \end{bmatrix}_{2 \times 2}, \mathbf{C}_2^A = [-0.3]_{1 \times 1}, \mathbf{D}_2^A = [0.3]_{1 \times 1}, \\ \boldsymbol{\beta}_1^R &= \begin{bmatrix} 1 & 0 \end{bmatrix}_{1 \times 2}, \mathbf{T}_1^R = \begin{bmatrix} -1 & 1 \\ 0 & -1 \end{bmatrix}_{2 \times 2}, \boldsymbol{\beta}_2^R = [1]_{1 \times 1}, \mathbf{T}_2^R = [-0.5]_{1 \times 1}, \\ \boldsymbol{\beta}_1^S &= \begin{bmatrix} 0.75 & 0.25 \end{bmatrix}_{1 \times 2}, \mathbf{T}_1^S = \begin{bmatrix} -1 & 0.25 \\ 0 & -0.25 \end{bmatrix}_{2 \times 2}, \boldsymbol{\beta}_2^S = [1]_{1 \times 1}, \mathbf{T}_2^S = [-0.5]_{1 \times 1} \end{aligned}$$

Hence,

$$\mathbf{T}_1^{R,0} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}_{2 \times 1}, \mathbf{T}_2^{R,0} = [0.5]_{1 \times 1}, \mathbf{T}_1^{S,0} = \begin{bmatrix} 0.75 \\ 0.25 \end{bmatrix}_{2 \times 1}, \mathbf{T}_2^{S,0} = [0.5]_{1 \times 1}.$$

Here, $\lambda_1 = 0.4$, $\lambda_2 = 0.3$, $\delta_1 = \delta_2 = 0.5$, $\mu_1 = 0.4$, $\mu_2 = 0.5$, $m_1^A = 2$, $m_2^A = 1$, $m_1^R = 2$, $m_2^R = 1$, $m_1^S = 2$, and $m_2^S = 1$. This is an 8-dimensional model with $m^R = 3$, $m^S = 3$, $b_1^R = 2$, $b_2^R = 4$, $b_1^S = 5$, $b_2^S = 7$, and $\mathbf{n}(\mathbf{x}) = x_6 + x_7 + x_8$. Therefore, the state space of the MC is given by $\mathcal{S} = \mathcal{S}^A \times \mathcal{S}^R \times \mathcal{S}^S$, where $\mathcal{S}^A = \{1, 2\} \times \{1\}$, $\mathcal{S}^R = \mathbb{Z}_{\geq 0}^3$, $\mathcal{S}^S = \{(0, 0, 0), (0, 0, 1), (0, 0, 2), (0, 1, 0), (0, 1, 1), (0, 2, 0), (1, 0, 0), (1, 0, 1), (1, 1, 0), (2, 0, 0)\}$.

6.2 Ergodicity condition

With the help of Lyapunov functions, we show that the inequality

$$\sum_{k=1}^K \frac{\lambda_k}{\mu_k} < c \tag{6.1}$$

is a necessary and sufficient ergodicity condition for the multi-class MAP/PH/ c queue with acyclic PH retrials considered. The Lyapunov function used to show the sufficiency of this condition will also aid us in finding a truncated state space with a given steady-state probability mass. When the arrival, service or retrial processes include phases, Lyapunov functions need to be chosen so that the phases

are carefully taken into consideration. Otherwise, a necessary and sufficient ergodicity condition may not be found. In order to obtain Lyapunov functions leading to such a condition, we start with Lyapunov functions that work for simple models, and then add terms to these functions for the additional complexities of the model.

The following two lemmas introduce two vectors which include variables that will be used in the additional terms.

Lemma 6. There exists a unique vector $\mathbf{u}_k \in \mathbb{R}^{m_k^A \times 1}$ for MAP with representation $(\mathbf{C}_k^A, \mathbf{D}_k^A)$ and $\lambda_k = \boldsymbol{\theta}_k^A \mathbf{D}_k^A \mathbf{e}$ such that $(\mathbf{C}_k^A + \mathbf{D}_k^A)\mathbf{u}_k = \lambda_k \mathbf{e} - \mathbf{D}_k^A \mathbf{e}$ and $\mathbf{u}_k \mathbf{e} = 1$.

Proof. The reduced linear system of equations $\hat{\mathbf{A}}_k \mathbf{u}_k = \hat{\mathbf{b}}_k$ with $\hat{\mathbf{A}}_k = \mathbf{M}_k(\mathbf{C}_k^A + \mathbf{D}_k^A) = (\mathbf{I}_{m_k^A} - \mathbf{e}_{m_k^A} \mathbf{e}_{m_k^A}^T)(\mathbf{C}_k^A + \mathbf{D}_k^A)$ and $\hat{\mathbf{b}}_k = \mathbf{M}_k(\lambda_k \mathbf{e} - \mathbf{D}_k^A \mathbf{e}) = (\mathbf{I}_{m_k^A} - \mathbf{e}_{m_k^A} \mathbf{e}_{m_k^A}^T)(\lambda_k \mathbf{e} - \mathbf{D}_k^A \mathbf{e})$, where $\mathbf{M}_k = \mathbf{I}_{m_k^A} + \mathbf{e}_{m_k^A}(-\mathbf{e}_{m_k^A}^T + \boldsymbol{\theta}_k^A)$ and $\mathbf{I}_{m_k^A}$ denotes the identity matrix of order m_k^A , has only its last equation altered to make the equation 0 on both sides. Because $(\mathbf{C}_k^A + \mathbf{D}_k^A)$ is an irreducible generator matrix, this implies $\hat{\mathbf{A}}_k$ is of rank $(m_k^A - 1)$. Thus, $(\mathbf{C}_k^A + \mathbf{D}_k^A)\mathbf{u}_k = \lambda_k \mathbf{e} - \mathbf{D}_k^A \mathbf{e}$ is consistent (see [116, Ch. 2.3]), and there exists a unique vector $\mathbf{u}_k$ under the normalization condition $\mathbf{u}_k \mathbf{e} = 1$. $\square$

Since transition rates describing arrivals in MAP depend on the phase of the process, elements of $\mathbf{u}_k$ will be used in additional terms to obtain an ergodicity condition based on average arrival rate λ_k instead of phase-dependent arrival rates in $\mathbf{D}_k^A$.

Lemma 7. There exists a unique vector $\mathbf{v}_k \in \mathbb{R}_{\geq 0}^{m_k^S \times 1}$ for PH service distribution with representation $(\boldsymbol{\beta}_k^S, \mathbf{T}_k^S)$ and $\mu_k = [-\boldsymbol{\beta}_k^S(\mathbf{T}_k^S)^{-1} \mathbf{e}]^{-1}$ such that $\mathbf{v}_k = -\mu_k(\mathbf{T}_k^S)^{-1} \mathbf{e}$ and $\boldsymbol{\beta}_k^S \mathbf{v}_k = 1$.

Proof. Matrix $-\mathbf{T}_k^S$ is a nonsingular M-matrix, $-(\mathbf{T}_k^S)^{-1} \geq 0$ [116, p. 626], and therefore, $\mathbf{v}_k \geq 0$ exists. This implies $\boldsymbol{\beta}_k^S \mathbf{v}_k = 1$ since $\boldsymbol{\beta}_k^S \mathbf{v}_k = -\mu_k(\boldsymbol{\beta}_k^S(\mathbf{T}_k^S)^{-1} \mathbf{e}) = -\mu_k(-\mu_k)^{-1}$. $\square$

Since transition rates describing PH service depend on the phase of the customer, elements of $\mathbf{v}_k$ will be used in additional terms to obtain an ergodicity condition based on average service rate μ_k instead of phase-dependent service rates in $\mathbf{T}_k^{S,0} = -\mathbf{T}_k^S \mathbf{e}$.

6.2.1 Necessary condition

The condition in (6.1) is necessary for ergodicity if it can be shown that the system is non-ergodic when the condition does not hold. Fortunately, the following theorem provides a non-ergodicity condition for MCs. Note that the theorem is originally proved for discrete-time MCs (see [12, p. 22] and [67, p. 30]), but here we give a continuous-time version which is obtained by considering the embedded MC.

Theorem 2. A MC with generator matrix $\mathbf{Q}$ is non-ergodic if there exists two constants $\tau, \sigma \in \mathbb{R}$ and a Lyapunov function $f : \mathcal{S} \rightarrow \mathbb{R}$ such that

- (i) $\sum_{\mathbf{y} \in \mathcal{S}} \mathbf{P}(\mathbf{x}, \mathbf{y}) |f(\mathbf{y}) - f(\mathbf{x})| \leq \tau$ for $\mathbf{x} \in \mathcal{S}$ and
- (ii) $\sum_{\mathbf{y} \in \mathcal{S}} \mathbf{P}(\mathbf{x}, \mathbf{y}) (f(\mathbf{y}) - f(\mathbf{x})) \geq 0$ for $\mathbf{x} \in \mathcal{R}$,

where the sets $\mathcal{R} = \{\mathbf{x} \in \mathcal{S} \mid f(\mathbf{x}) > \sigma\}$ and $(\mathcal{S} \setminus \mathcal{R})$ are non-empty, and the matrix $\mathbf{P} \in \mathbb{R}_{\geq 0}^{|\mathcal{S}| \times |\mathcal{S}|}$ is given elementwise as

$$\mathbf{P}(\mathbf{x}, \mathbf{y}) = \begin{cases} \mathbf{Q}(\mathbf{x}, \mathbf{y}) / |\mathbf{Q}(\mathbf{x}, \mathbf{x})| & \text{if } \mathbf{y} \neq \mathbf{x} \\ 0 & \text{otherwise} \end{cases} \quad \text{for } \mathbf{x}, \mathbf{y} \in \mathcal{S}.$$

This theorem is used to prove a non-ergodicity condition for an M/M/c retrial queue with exponential retrial in [66, p. 98]. Therein, a Lyapunov function $f(\mathbf{x})$ linear in the infinite variables (i.e., numbers of different class customers in the orbits) is used. We have also chosen a linear Lyapunov function, but have added constant terms including elements of vectors $\mathbf{u}_k$ and $\mathbf{v}_k$ from Lemmas 6 and 7. Thus, we consider

$$f(\mathbf{x}) = \sum_{k=1}^K \frac{1}{\mu_k} \left(\sum_{i=1}^{m_k^R} x_{b_k^R+i} \right) + \sum_{k=1}^K \frac{\mathbf{u}_k(x_k)}{\mu_k} + \sum_{k=1}^K \frac{1}{\mu_k} \left(\sum_{i=1}^{m_k^S} \mathbf{v}_k(i) x_{b_k^S+i} \right).$$

The first term of $f(\mathbf{x})$ is the initial function, whereas the other two terms are added to obtain a phase-independent condition. Note that each of the three terms is in the form of a summation of K other terms each corresponding to a different customer class.

Since $\mathbf{Q}$ is a generator matrix, $\mathbf{P}(\mathbf{x}, \mathbf{y}) \leq 1$ for $\mathbf{x}, \mathbf{y} \in \mathcal{S}$. The value of $|f(\mathbf{y}) - f(\mathbf{x})|$ is finite if $\mathbf{P}(\mathbf{x}, \mathbf{y}) > 0$ for $\mathbf{x}, \mathbf{y} \in \mathcal{S}$. Besides, each row of the matrix $\mathbf{P}$ includes a finite number of nonzero elements due to the form of the particular model considered. Therefore, there exists some constant τ such that condition (i) is satisfied.

After some algebraic operations, we obtain

$$\frac{1}{|\mathbf{Q}(\mathbf{x}, \mathbf{x})|} \sum_{\mathbf{y} \in \mathcal{S}} \mathbf{Q}(\mathbf{x}, \mathbf{y})(f(\mathbf{y}) - f(\mathbf{x})) = \frac{1}{|\mathbf{Q}(\mathbf{x}, \mathbf{x})|} \left(\sum_{k=1}^K \frac{\lambda_k}{\mu_k} - \mathbf{n}(\mathbf{x}) \right).$$

The right-hand side of this equation is nonnegative if $\sum_{k=1}^K (\lambda_k / \mu_k) \geq c$ since $|\mathbf{Q}(\mathbf{x}, \mathbf{x})| > 0$ and $\mathbf{n}(\mathbf{x}) \leq c$ hold for $\mathbf{x} \in \mathcal{S}$. Besides, sets $\mathcal{R}$ and $(\mathcal{S} \setminus \mathcal{R})$ are non-empty when $\sigma = \max_{\mathbf{x} \in \mathcal{S}} \left(\sum_{k=1}^K \mathbf{u}_k(x_k) / \mu_k \right)$. Therefore, condition (ii) also holds.

6.2.2 Sufficient condition

In coming up with a Lyapunov function to show the sufficiency of the condition in (6.1) for ergodicity, we benefit from the next theorem in [148]. Note that the Lyapunov function is assumed to be nonnegative in [148], but the theorem is also valid if the Lyapunov function is bounded from below (see [66, p. 97]). This theorem will also be used to bound the steady-state probability mass associated with the truncated state space from below as proposed in [44].

Theorem 3. The MC is ergodic if and only if there exists a Lyapunov function $g : \mathcal{S} \rightarrow \mathbb{R}$ that is bounded from below and a finite set $\mathcal{C} \subset \mathcal{S}$ satisfying the three conditions

- (i) $\{\mathbf{x} \in \mathcal{S} \mid g(\mathbf{x}) \leq \tau\}$ is finite for all $\tau < \infty$,
- (ii) $\phi(\mathbf{x}) \leq -\gamma$ for all $\mathbf{x} \in \mathcal{S} \setminus \mathcal{C}$ and some $\gamma > 0$, and
- (iii) $\phi(\mathbf{x}) < \infty$ for all $\mathbf{x} \in \mathcal{C}$,

where $\phi(\mathbf{x}) = \sum_{\mathbf{y} \in \mathcal{S}} \mathbf{Q}(\mathbf{x}, \mathbf{y}) (g(\mathbf{y}) - g(\mathbf{x}))$ is called the drift in state $\mathbf{x} \in \mathcal{S}$.

We will be choosing a quadratic Lyapunov function $g(\mathbf{x})$ that is similar to those discussed in [30] to obtain a drift function $\phi(\mathbf{x})$ whose infinite variables have negative coefficients for $\mathbf{x} \in \mathcal{S}$. If customers in the orbits attempt to receive service in all PH retrial phases, adding terms with variables each corresponding to the number of customers in a service phase leads to the condition in (6.1). However, if the PH retrial process of a class k customer includes a phase, say i , in which no attempt is made to receive service (i.e., if $\mathbf{T}_k^{R,0}(i) = 0$, where $\mathbf{T}_k^{R,0} = -\mathbf{T}_k^R \mathbf{e}$), then the coefficient of the corresponding variable, $x_{b_k^R+i}$, is positive in $\phi(\mathbf{x})$ at states with no busy servers. Therefore, the value of $g(\mathbf{x})$ should also depend on the number of customers in retrial phases. This can be managed by adding carefully chosen terms so that $\phi(\mathbf{x})$ at states with no busy servers become negative for a sufficiently large number of customers in the orbits. Based on this explanation, the following lemma gives the vector that includes variables to be used in the term added due to PH retrials of class k customers.

Lemma 8. For acyclic PH retrial distribution with representation $(\beta_k^R, \mathbf{T}_k^R)$, let $\hat{\mathbf{U}}_k = \mathbf{T}_k^R + \text{diag}(\mathbf{T}_k^{R,0})$ and $\boldsymbol{\eta}_k \in \mathbb{R}^{m_k^R \times 1}$ be given elementwise as

$$\boldsymbol{\eta}_k(i) = \begin{cases} -c/\mu_k & \text{if } i \in \mathcal{I}_k \\ 0 & \text{otherwise} \end{cases} \quad \text{for } i = 1, \dots, m_k^R,$$

where $\mathcal{I}_k = \{i \in \{1, \dots, m_k^R\} \mid \mathbf{T}_k^{R,0}(i) = 0\}$. Then there exists $\mathbf{w}_k \in \mathbb{R}^{m_k^R \times 1}$ such that $\hat{\mathbf{U}}_k \mathbf{w}_k = \boldsymbol{\eta}_k$.

Proof. Note that all elements in row i of $\hat{\mathbf{U}}_k$ are zero if and only if $\hat{\mathbf{U}}_k(i, i) = 0$. Then the result follows from the row echelon form of upper-triangular matrix $\hat{\mathbf{U}}_k$ obtained by interchanging all zero rows with nonzero rows below them (see [116, Ch. 2.1]), that its rank is equal to its number of nonzero rows, and $\hat{\mathbf{U}}_k \mathbf{w}_k = \boldsymbol{\eta}_k$ is consistent and has infinitely many solutions (see [116, Ch. 2.3]). Note that

negative elements of $\boldsymbol{\eta}_k$ could take a smaller value than $-c/\mu_k$ that may lead to a smaller truncated state space. We choose this value in order to bound coefficients of infinite variables from above by $(\sum_{k=1}^K(\lambda_k/\mu_k) - c)$ in $\phi(\mathbf{x})$ for all states. $\square$

There exist infinitely many solutions to $\hat{\mathbf{U}}_k \mathbf{w}_k = \boldsymbol{\eta}_k$. Hence, additional constraints need to be imposed. Here, we choose to set $\mathbf{w}_k(i)$ to 1 if row i of $\hat{\mathbf{U}}_k$ is zero. The elements of vector $\boldsymbol{\eta}_k$ contribute to coefficients of infinite variables corresponding to class- k retrial customers in $\phi(\mathbf{x})$; hence, its elements need to be nonpositive with at least one negative element when $\mathcal{I}_k \neq \emptyset$. If PH retrials are allowed to be non-acyclic, it is a possibility that $\hat{\mathbf{U}}_k \mathbf{w}_k = \boldsymbol{\eta}_k$ is inconsistent since $\hat{\mathbf{U}}_k$ is a singular M-matrix. Hence, the acyclicity assumption is necessary due to the form of the chosen $g(\mathbf{x})$. We conjecture that the condition in (6.1) is also necessary and sufficient when PH retrials are non-acyclic; however, we have not managed to provide a suitable $g(\mathbf{x})$ for such a system.

Having defined all necessary variables, as the Lyapunov function we consider

$$\begin{aligned} g(\mathbf{x}) = & \sum_{k'=1}^K \sum_{l'=1}^K \frac{1}{2\mu_{k'}\mu_{l'}} \left(\sum_{i=1}^{m_{k'}^R} x_{b_{k'}^R+i} \right) \left(\sum_{i=1}^{m_{l'}^R} x_{b_{l'}^R+i} \right) + \sum_{k'=1}^K \left(\sum_{l'=1}^K \frac{\mathbf{u}_{l'}(x_{l'})}{\mu_{k'}\mu_{l'}} \right) \left(\sum_{i=1}^{m_{k'}^R} x_{b_{k'}^R+i} \right) \\ & + \sum_{k'=1}^K \left(\sum_{l'=1}^K \frac{1}{\mu_{k'}\mu_{l'}} \left(\sum_{j=1}^{m_{l'}^S} \mathbf{v}_{l'}(j) x_{b_{l'}^S+j} \right) \right) \left(\sum_{i=1}^{m_{k'}^R} x_{b_{k'}^R+i} \right) \\ & + \sum_{k'=1}^K \sum_{i=1}^{m_{k'}^R} \mathbf{w}_{k'}(i) x_{b_{k'}^R+i} + \sum_{k'=1}^K \alpha_{k'}(\mathbf{x}) \left(\sum_{i=1}^{m_{k'}^S} x_{b_{k'}^S+i} \right), \end{aligned}$$

where

$$a_k(i, \mathbf{x}) = \mathbf{w}_k(i) + \frac{1}{2\mu_k^2} + \sum_{k'=1}^K \frac{\mathbf{u}_{k'}(x_{k'})}{\mu_k\mu_{k'}} + \sum_{k'=1}^K \frac{1}{\mu_k\mu_{k'}} \left(\sum_{j'=1}^{m_{k'}^S} \mathbf{v}_{k'}(j') x_{b_{k'}^S+j'} \right)$$

for $i \notin \mathcal{I}_k$ and $\alpha_k(\mathbf{x}) = \min_{i \notin \mathcal{I}_k} \left(a_k(i, \mathbf{x}) - c/(\mathbf{T}_k^{R,0}(i)\mu_k) \right)$ for $\mathbf{x} \in \mathcal{S}$ and $k = 1, \dots, K$. The first term of $g(\mathbf{x})$ is the initial quadratic Lyapunov function. The second and third terms are added to obtain a phase-independent condition, and the last two terms are added due to PH retrials. Note that, $\alpha_k(\mathbf{x})$ is well-defined since $\mathcal{I}_k \neq \{1, \dots, m_k^R\}$ for $k = 1, \dots, K$ due to Definitions 7 and 8. Then

$g(\mathbf{x})$ is a quadratic polynomial in which the coefficients of all infinite variables are finite and the coefficient of each quadratic term $(x_{b_k^R+i}x_{b_{k'}^R+i'})$ is positive for $i = 1, \dots, m_k^R$, $i' = 1, \dots, m_{k'}^R$, and $k, k' = 1, \dots, K$. Hence, the function g is bounded from below and condition (i) holds.

For $\mathbf{n}(\mathbf{x}) = c$, the drift is given by

$$\begin{aligned} \phi(\mathbf{x}) = & \sum_{k=1}^K \frac{1}{\mu_k} \left(\sum_{k'=1}^K \frac{\lambda_{k'}}{\mu_{k'}} - c \right) \left(\sum_{i=1}^{m_k^R} x_{b_k^R+i} \right) - \sum_{k=1}^K \frac{c}{\mu_k} \left(\sum_{i \in \mathcal{I}_k} x_{b_k^R+i} \right) \\ & + \sum_{k=1}^K \sum_{i=1}^{m_k^A} D_k^A(x_k, i) \left[\frac{\mathbf{u}_k(i)}{\mu_k^2} + \sum_{\substack{k'=1 \\ k' \neq k}}^K \frac{\mathbf{u}_{k'}(x_{k'})}{\mu_k \mu_{k'}} \right] \\ & + \sum_{k=1}^K \sum_{i=1}^{m_k^A} D_k^A(x_k, i) \left[\sum_{k'=1}^K \frac{1}{\mu_k \mu_{k'}} \left(\sum_{j'=1}^{m_{k'}^S} \mathbf{v}_{k'}(j') x_{b_{k'}^S+j'} \right) \right] \\ & + \sum_{k=1}^K \sum_{i=1}^{m_k^A} D_k^A(x_k, i) \left[\left(\sum_{j=1}^{m_k^R} \beta_k^R(j) \mathbf{w}_k(j) \right) + \frac{1}{2\mu_k^2} \right] \\ & - \sum_{k=1}^K \alpha_k(\mathbf{x}) \left(\sum_{i=1}^{m_k^S} x_{b_k^S+i} \mathbf{T}_k^{S,0}(i) \right); \end{aligned}$$

on the other hand, for $\mathbf{n}(\mathbf{x}) < c$ the drift is given by

$$\begin{aligned} \phi(\mathbf{x}) = & \sum_{k=1}^K \frac{1}{\mu_k} \left(\sum_{k'=1}^K \frac{\lambda_{k'}}{\mu_{k'}} - \sum_{k'=1}^K \sum_{i=1}^{m_{k'}^S} x_{b_{k'}^S+i} \right) \left(\sum_{i=1}^{m_k^R} x_{b_k^R+i} \right) - \sum_{k=1}^K \frac{c}{\mu_k} \left(\sum_{i \in \mathcal{I}_k} x_{b_k^R+i} \right) \\ & + \sum_{k=1}^K \sum_{i \notin \mathcal{I}_k} x_{b_k^R+i} \mathbf{T}_k^{R,0}(i) (-a_k(i, \mathbf{x}) + \alpha_k(\mathbf{x})) \\ & + \sum_{k=1}^K \alpha_k(\mathbf{x}) \left(\sum_{i=1}^{m_k^A} D_k^A(x_k, i) - \sum_{i=1}^{m_k^S} x_{b_k^S+i} \mathbf{T}_k^{S,0}(i) \right) \quad \text{for } \mathbf{x} \in \mathcal{S}. \end{aligned}$$

When $\sum_{k=1}^K (\lambda_k/\mu_k) < c$ holds, coefficients of all infinite variables are negative; hence, condition (iii) holds. Besides, $\mathcal{C} = \{\mathbf{x} \in \mathcal{S} \mid \phi(\mathbf{x}) > -\gamma\}$ is finite for any arbitrarily chosen $\gamma > 0$. Therefore, condition (ii) also holds and the model is ergodic if $\sum_{k=1}^K (\lambda_k/\mu_k) < c$. As discussed in [44], for some given positive $\epsilon < 1$ we have $\sum_{\mathbf{x} \in \mathcal{C}} \boldsymbol{\pi}(\mathbf{x}) \geq 1 - \epsilon$, where $\gamma = \sup_{\mathbf{x} \in \mathcal{S}} \phi(\mathbf{x}) (1/\epsilon - 1)$ and $\boldsymbol{\pi}$ is the steady-state solution.

Example 4. (cont'd) In this example,

$$\mathbf{u}_1 = \begin{bmatrix} 0.25 \\ 0.75 \end{bmatrix}_{2 \times 1}, \quad \mathbf{v}_1 = \begin{bmatrix} 0.8 \\ 1.6 \end{bmatrix}_{2 \times 1}, \quad \mathbf{w}_1 = \begin{bmatrix} 6 \\ 1 \end{bmatrix}_{2 \times 1}, \quad \mathbf{u}_2 = \mathbf{v}_2 = \mathbf{w}_2 = \begin{bmatrix} 1 \end{bmatrix}_{1 \times 1}.$$

Then $\mathcal{I}_1 = \{1\}$, $\mathcal{I}_2 = \emptyset$,

$$a_1(2, \mathbf{x}) = 4.125 + 6.25\mathbf{u}_1(x_1) + 5\mathbf{u}_2(x_2) + 5x_6 + 10x_7 + 5x_8, \quad \alpha_1(\mathbf{x}) = a_1(2, \mathbf{x}) - 5,$$

$$a_2(1, \mathbf{x}) = 3 + 5\mathbf{u}_1(x_1) + 4\mathbf{u}_2(x_2) + 4x_6 + 8x_7 + 4x_8, \quad \alpha_2(\mathbf{x}) = a_2(1, \mathbf{x}) - 8.$$

The Lyapunov function is given by

$$\begin{aligned} g(\mathbf{x}) = & 3.125(x_3 + x_4)^2 + 5(x_3 + x_4)x_5 + 2x_5^2 + 6x_3 + x_4 + x_5 \\ & + (6.25\mathbf{u}_1(x_1) + 5\mathbf{u}_2(x_2) + 5x_6 + 10x_7 + 5x_8)(x_3 + x_4) \\ & + (5\mathbf{u}_1(x_1) + 4\mathbf{u}_2(x_2) + 4x_6 + 8x_7 + 4x_8)x_5 + \alpha_1(\mathbf{x})(x_6 + x_7) + \alpha_2(\mathbf{x})x_8. \end{aligned}$$

For $\mathbf{n}(\mathbf{x}) = 2$, the drift is given by

$$\begin{aligned} \phi(\mathbf{x}) = & -6x_3 - x_4 - 0.8x_5 + \mathbb{1}_{x_1=2}(4\mathbf{u}_2(x_2) + 4x_6 + 8x_7 + 4x_8 + 8.55) \\ & + (1.5\mathbf{u}_1(x_1) + 1.2x_6 + 2.4x_7 + 1.2x_8 + 2.1) \\ & + \alpha_1(\mathbf{x})(-0.75x_6 - 0.25x_7) + \alpha_2(\mathbf{x})(-0.5x_8), \end{aligned}$$

where $\mathbb{1}$ denotes the indicator function, and for $\mathbf{n}(\mathbf{x}) < 2$, the drift is given by

$$\begin{aligned} \phi(\mathbf{x}) = & 2.5(-0.4 - x_6 - x_7 - x_8)(x_3 + x_4) + 2(-0.4 - x_6 - x_7 - x_8)x_5 \\ & + \alpha_1(\mathbf{x})(\mathbb{1}_{x_1=2} 0.8 - 0.75x_6 - 0.25x_7) + \alpha_2(\mathbf{x})(0.3 - 0.5x_8). \end{aligned}$$

6.3 Numerical results

Once a Kronecker representation of the truncated generator matrix $\bar{\mathbf{Q}}$ is obtained as in [30, 46], one can employ a memory-efficient Kronecker-based iterative solver in which the nonzeros of $\bar{\mathbf{Q}}$ are not stored and no factorization takes place during the course of computing $\bar{\pi}$. In practice, this approach always saves a significant amount of memory. Another approach is to employ an LDQBD model, where level l has the state space

$$\mathcal{S}(l) = \{\mathbf{x} = (x_1, \dots, x_{K+m^R+m^S}) \in \mathcal{S} \mid l = \max\{x_{K+1}, \dots, x_{K+m^R}\}\},$$

and then use the matrix-analytic method proposed in [18]. In this method, conditional expected sojourn time matrices between two given truncation levels need to be computed and stored. For our system, the conditional expected sojourn time matrix at level l includes about $(\bar{M}^2 l^{2(m^R-1)})$ nonzeros, where $\bar{M} = |\mathcal{S}|(\prod_{k=1}^K m_k^A)m^R$. Hence, the memory allocated to store these matrices becomes extremely large. Phung-Duc et al. [128] propose an algorithm to compute the conditional expected sojourn time matrices of an LDQBD with a smaller memory usage (see [128, Algorithm. 1]). However, the stationary distribution is computed using the algorithm proposed in [18] (see [128, Algorithm. 3]) which requires computing and storing the conditional expected sojourn time matrices (see [128, Remark. 3.5]). Hence, memory usage is expected to be large in this algorithm. On the other hand, the Kronecker-based iterative solver is expected to solve the truncated system with comparable accuracy if the value of the stopping tolerance it uses is sufficiently small.

We implemented a Kronecker solver [52] built on the Nsolve package [23] of the APNN toolbox [14]. The solver obtains the truncated state space of the model, generates the Kronecker structured matrix of the truncated model, and computes its steady-state solution by using the SOR method of Nsolve. All experiments are carried out on an Intel Core i7 Quad-Core 3.6 GHz processor with 32 GB of main memory. Iterations are stopped when infinity norm of the residual vector of the truncated model (i.e., $\|\bar{\pi}\bar{Q}\|_\infty$) becomes smaller than 10^{-15} , and the relaxation parameter of SOR is chosen as 0.9.

We considered six different models with $\epsilon = 0.2$. Table 6.1 reports the results of numerical experiments with these models. The first column gives the name of the model. ERL₁ is the model introduced in Example 4. ERL₂ and ERL₃ are obtained by multiplying the matrices describing the arrival processes of ERL₁ by 0.75 and 0.5, respectively. EXP₁ differs from the first model in that the retrial time of customer class 1 is exponentially distributed with average retrial rate 0.5. EXP₂ and EXP₃ are obtained by multiplying the matrices describing the arrival processes of EXP₁ by 0.75 and 0.5, respectively. The second column gives the traffic intensity $\rho = (\sum_{k=1}^K \lambda_k/\mu_k)c^{-1}$. The third column gives the number of states in the truncated state space. Columns $E_{1,1}$, $E_{1,2}$, and $E_{2,1}$ give the average

number of class 1 customers in retrial phases 1 and 2, and the average number of class 2 customers in retrial phase 1, respectively. Retrial process of customer class 1 has a single phase in EXP_1 , EXP_2 , and EXP_3 ; hence, $E_{1,2}$ is undefined for those models. Column P_{block} provides the probability that an arriving customer finds all servers busy. Column $\|\bar{\pi}\bar{Q}\|_1$ provides the 1-norm of the residual vector of the truncated model which is an indicator of the accuracy of the solution to the truncated model. Column $\|\tilde{\pi}Q\|_1$ provides the 1-norm of the residual vector of the infinite model and is obtained from

$$\|\tilde{\pi}Q\|_1 = \sum_{x \in \mathcal{S}} |r(x) - \sum_{y \notin \mathcal{S}} \bar{\pi}(x)Q(x, y)| + \sum_{x \in \mathcal{S}} \sum_{y \notin \mathcal{S}} \bar{\pi}(x)Q(x, y) \quad \text{with } r = \bar{\pi}\bar{Q}.$$

This value is an indicator of the accuracy of the solution to the infinite model.

Table 6.1: Numerical results

Model	ρ	$ \bar{\mathcal{S}} $	$E_{1,1}$	$E_{1,2}$	$E_{2,1}$	P_{block}	$\ \bar{\pi}\bar{Q}\ _1$	$\ \tilde{\pi}Q\ _1$
ERL ₁	0.8	29,932,260	0.2584	2.0165	2.8985	0.6784	$9e-14$	$9e-14$
ERL ₂	0.6	811,800	0.1097	0.4422	0.6756	0.4164	$2e-14$	$2e-12$
ERL ₃	0.4	12,800	0.0317	0.0860	0.1511	0.2048	$6e-15$	$7e-6$
EXP ₁	0.8	270,400	3.1663		2.5486	0.6755	$8e-14$	$8e-14$
EXP ₂	0.6	25,600	0.7154		0.6302	0.4142	$2e-14$	$2e-11$
EXP ₃	0.4	1,600	0.1464		0.1461	0.2038	$4e-15$	$6e-6$

Relative difference between average numbers of retrial customers in ERL_i and EXP_i becomes relatively large as traffic intensity increases. This difference ranges from 0.03 (customer class 2 in ERL_3 and EXP_3) to 0.39 (customer class 1 in ERL_1 and EXP_1). Relative difference between blocking probabilities is around 0.005. In ERL_1 and EXP_1 , residuals of truncated and infinite models are about the same; hence, truncation error is not larger than numerical error. In other models, truncation error is larger than numerical error. Therefore, as traffic intensity increases, choosing a smaller ϵ value does not introduce additional inaccuracy to the computed solution.

Chapter 7

Conclusion

Markov chains (MCs) are state- and transition-based probabilistic models widely used to analyze the behavior of systems arising in various application areas. Most realistic systems are composed of interacting subsystems and a countably infinite multi-dimensional state representation in which each subsystem typically corresponds to a different dimension. There are different numerical techniques for the transient and steady-state analysis of finite MCs. When the state space is countably infinite, an MC obtained by appropriately truncating the countably infinite state space, can be analyzed. The truncated state space needs to be chosen carefully so that the state truncation error is kept small and the truncated state space does not include many states with small probability mass. In the steady-state analysis, the existence of the limiting distribution depends on the model parameters. Hence, the existence of a steady-state solution needs to be shown before analyzing the system.

In multi-dimensional systems, the state truncation error is usually kept small only if the truncated state space is chosen large. When the truncated system is chosen large enough, the memory requirement for storing the generator matrix becomes significant. Using Kronecker products, it is possible to represent generator matrices underlying multi-dimensional MCs compactly. When the reachable state space is significantly smaller than the product state space, inefficiency due

to unreachable states can be avoided by letting the set of rows and the set of columns equal to the reachable state space. Compact storage of the generator matrix underlying an MC with no unreachable states and efficient implementation of analysis methods using Kronecker operations require partitioning the reachable state space into Cartesian products of subsets of subsystem state spaces. For practical purposes that aid the use of Kronecker operations, we are interested in minimizing the number of partitions. We show that partitioning a d -dimensional reachable state space into minimum number of Cartesian products of subsets of subsystem state spaces is NP-complete for $d > 2$, and we propose two algorithms to automate the partitioning of an arbitrary reachable state space.

Kronecker representation of the generator matrix helps decreasing the memory requirement in numerical analysis. However, analysis methods based on vector-Kronecker product multiplication need to be employed when the generator matrix underlying the MC is represented compactly using Kronecker products. When the factors in the Kronecker product terms are relatively dense, vector-Kronecker product multiplication can be performed efficiently by the shuffle algorithm. When the factors are relatively sparse, it may be more efficient to obtain nonzero elements of the generator matrix in Kronecker form on-the-fly and multiply them with corresponding elements of the vector. We propose a modification to the shuffle algorithm that multiplies relevant elements of the vector with submatrices of factors in which zero rows and columns are omitted. This approach avoids unnecessary floating-point operations that evaluate to zero during the course of the multiplication. Numerical experiments on a large number of models indicate that, in many cases, the modified shuffle algorithm requires less time and less floating-point operations than the other algorithms. Furthermore, the modification of the shuffle algorithm decreases the memory requirement in all but two models.

Time evolution of systems of stochastic chemical kinetics can be described by a set of ordinary differential equations. We solve this ODE system by the BDF5 (backward differentiation formulae method of order 5) method with vectors in hierarchical Tucker decomposition (HTD) format. A naive approach is to truncate the vectors using a fixed truncation error tolerance and fixed error bound during

the computation. However, these fixed truncation strategies causes the ranks to become large during the computation and the computation of the transient probability distribution at a given time instant becomes impossible for some models. Using adaptive truncation strategies, we significantly reduce the ranks of the solution vectors as well as time and memory requirements for the problems for which the fixed truncation strategies do not work. Numerical experiments indicate that solvers with vectors in HTD format become time-wise and memory-wise efficient compared to the solvers with flat vectors as the number of dimensions increases.

Besides an application of the transient analysis of a multi-dimensional MC, we are interested in the steady-state solution of a multi-class MAP/PH/ c queue with acyclic PH retrials. Necessary and sufficient conditions for the ergodicity of this system is obtained from criteria based on mean drifts. With the help of an appropriately chosen Lyapunov function, the countably infinite state space is truncated. Then the truncated system is represented as a multi-dimensional MC and a Kronecker representation of its underlying infinitesimal generator matrix is given. Finally, the truncated system is solved using the SOR method and an approximation to the steady-state solution of the system is computed. The largeness of the state space of a multi-dimensional MC is a major difficulty in numerical techniques. The work in this thesis proposes that compact representation of the solution vectors is a promising approach to analyze larger and more realistic systems numerically. However, there are things to be done. The compact vectors have to be used in more advanced solution techniques like projection or multi-level solution techniques. Parallelization of the solution techniques may also improve time requirement significantly. Besides, different approaches with adaptive binary tree structures and adaptive state space partitioning are worth investigation.

Bibliography

- [1] APNN-Toolbox, Abstract Petri Net Notation Toolbox. <http://www4.cs.uni-dortmund.de/APNN-TOOLBOX>. 2004.
- [2] Netlib. A Collection of Mathematical Software, Papers, and Databases. <http://www.netlib.org>. 2016.
- [3] TT-Toolbox, Tensor Train Toolbox. <https://github.com/oseledets/TT-Toolbox>. 2014.
- [4] G. M. Adelson-Velskii and E. M. Landis. An algorithm for the organization of information. *Soviet Mathematics Doklady*, 3:1259–1263, 1962.
- [5] M. Ajmone Marsan, S. Donatelli, and F. Neri. GSPN models of Markovian multiserver multiqueue systems. *Performance Evaluation*, 11:227–240, 1990.
- [6] J. R. Artalejo. Accessible bibliography on retrial queues. *Mathematical and Computer Modelling*, 30:1–6, 1999.
- [7] J. R. Artalejo. Accessible bibliography on retrial queues: Progress in 2000–2009. *Mathematical and Computer Modelling*, 51:1071–1081, 2010.
- [8] J. R. Artalejo and A. Gómez-Corral. Modelling communication systems with phase type service and retrial times. *IEEE Communications Letters*, 11:955–957, 2007.
- [9] J. R. Artalejo and A. Gómez-Corral. *Retrial Queueing Systems: A Computational Approach*. Springer, Berlin, 2008.

- [10] J. R. Artalejo and T. Phung-Duc. Markovian retrial queues with two way communication. *Journal of Industrial and Management Optimization*, 8:781–806, 2012.
- [11] U. M. Ascher and L. R. Petzold. *Computer methods for ordinary differential equations and differential-algebraic equation*. SIAM, Philadelphia, 1998.
- [12] S. Asmussen. *Applied Probability and Queues*. Springer, New York, 2003.
- [13] K. Avrachenkov, E. Morozov, and B. Steyaert. Sufficient stability conditions for multi-class constant retrial rate systems. *Queueing Systems*, 82:1–23, 2015.
- [14] F. Bause, P. Buchholz, and P. Kemper. A toolbox for functional and quantitative analysis of DEDS. In R. Puigjaner, N. N. Savino, and B. Serra, editors, *International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, volume 1469 of *Lecture Notes in Computer Science*, pages 356–359. Springer, Heidelberg, Germany, 1998.
- [15] S. L. Bell and R. J. Williams. Dynamic scheduling of a system with two parallel servers in heavy traffic with resource pooling: asymptotic optimality of a threshold policy. *The Annals of Applied Probability*, 11(3):608–649, 2001.
- [16] O. Bournez, O. Maler, and A. Pnueli. Orthogonal Polyhedra: Representation and Computation. In *Hybrid Systems: Computation and Control*, volume 1569 of *Lecture Notes in Computer Science*, pages 46–60. Springer, 1999.
- [17] L. Breuer, A. Dudin, and V. Klimenok. A retrial BMAP/PH/N system. *Queueing Systems*, 40:433–457, 2002.
- [18] L. Bright and P. G. Taylor. Calculating the equilibrium distribution in level dependent quasi-birth-and-death processes. *Stochastic Models*, 11:497–525, 1995.
- [19] P. Buchholz. A class of hierarchical queueing networks and their analysis. *Queueing Systems*, 15(1):59–80, 1994.

- [20] P. Buchholz. Hierarchical structuring of superposed GSPNs. *IEEE Transactions on Software Engineering*, 25(2):166–181, 1999.
- [21] P. Buchholz. Structured analysis approaches for large Markov chains. *Applied Numerical Mathematics*, 31(4):375–404, 1999.
- [22] P. Buchholz. Adaptive decomposition and approximation for the analysis of stochastic Petri nets. *Performance Evaluation*, 56:23–52, 2004.
- [23] P. Buchholz. The Nsolve package. <http://ls4-www.cs.tu-dortmund.de/download/buchholz/struct-matrix-market.html>, 2015.
- [24] P. Buchholz, G. Ciardo, S. Donatelli, and P. Kemper. Complexity of memory-efficient Kronecker operations with applications to the solution of Markov models. *INFORMS Journal on Computing*, 12(3):203–222, 2000.
- [25] P. Buchholz and T. Dayar. Block SOR for Kronecker structured representations. *Linear Algebra and its Applications*, 386:83–109, 2004.
- [26] P. Buchholz and T. Dayar. Comparison of multilevel methods for Kronecker-based Markovian representations. *Computing*, 73(4):349–371, 2004.
- [27] P. Buchholz and T. Dayar. Block SOR preconditioned projection methods for Kronecker structured Markovian representations. *SIAM Journal on Scientific Computing*, 26(4):1289–1313, 2005.
- [28] P. Buchholz and T. Dayar. On the convergence of a class of multilevel methods for large sparse Markov chains. *SIAM Journal on Matrix Analysis and Applications*, 29(3):1025–1049, 2007.
- [29] P. Buchholz, T. Dayar, J. Kriege, and M. C. Orhan. Compact representation of solution vectors in Kronecker-based Markovian analysis. In G. Agha and B. V. Houdt, editors, *Proceedings of the 13th International Conference on Quantitative Evaluation of Systems*, volume 9826 of *Lecture Notes in Computer Science*, pages 260–276. Springer, Heidelberg, 2016.

- [30] P. Buchholz, T. Dayar, M. C. Orhan, and W. Sandmann. On the numerical solution of Kronecker-based infinite level-dependent QBD processes. *Performance Evaluation*, 70:663–681, 2013.
- [31] P. Buchholz and P. Kemper. Compact representations of probability distributions in the analysis of superposed GSPNs. In *Proceedings of the 9th International Workshop on Petri Nets and Performance Models, Aachen, Germany*, pages 81–90. IEEE, 2001.
- [32] P. Buchholz, J. Kriege, and I. Felko. *Input Modeling with Phase-Type Distributions and Markov Models: Theory and Applications*. Springer, Heidelberg, 2014.
- [33] K. Burrage, M. Hegland, S. Macnamara, and R. Sidje. A Krylov-based finite state projection algorithm for solving the chemical master equation arising in the discrete modelling of biological systems. In A. N. Langville and W. J. Stewart, editors, *Proceedings of the A. A. Markov 150th Anniversary Meeting*, pages 21–38. Boson Books, 2006.
- [34] Y. Cao, D. T. Gillespie, and L. R. Petzold. The slow-scale stochastic simulation algorithm. *The Journal of Chemical Physics*, 122(1):014116, 2005.
- [35] J. D. Carroll and J. J. Chang. Analysis of individual differences in multidimensional scaling via an n-way generalization of “Eckart-Young” decomposition. *Psychometrika*, 35(3):283–319, 1970.
- [36] S. R. Chakravorthy. Analysis of MAP/PH/c Retrieval Queue with Phase Type Retrievals – Simulation Approach. In A. Dudin, V. Klimenok, G. Tsarenkov, and S. Dudin, editors, *Modern Probabilistic Methods for Analysis of Telecommunication Networks*, volume 356 of *Communications in Computer and Information Science*, pages 37–49. Springer, Berlin, 2013.
- [37] D. Cheng, H. Qi, and Y. Zhao. *An introduction to semi-tensor product of matrices and its applications*. World Scientific, Singapore, 2012.
- [38] G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad. Stochastic well-formed colored nets and symmetric modeling applications. *IEEE Transactions on Computers*, 42:1343–1360, 1993.

- [39] B. D. Choi and Y. Chang. MAP1,MAP2/M/c retrial queue with the retrial group of finite capacity and geometric loss. *Mathematical and Computer Modelling*, 30:99–113, 1999.
- [40] B. D. Choi, Y. Chang, and B. Kim. MAP1,MAP2/M/c retrial queue with guard channels and its application to cellular networks. *TOP*, 7:231–248, 1999.
- [41] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT press, 3rd edition, 2009.
- [42] M. Davio. Kronecker products and shuffle algebra. *IEEE Transactions on Computers*, 30(2):116–125, 1981.
- [43] T. Dayar. *Analyzing Markov Chains Using Kronecker Products: Theory and Applications*. Springer, New York, 2012.
- [44] T. Dayar, H. Hermanns, D. Spieler, and V. Wolf. Bounding the equilibrium distribution of Markov population models. *Numerical Linear Algebra with Applications*, 18:931–946, 2011.
- [45] T. Dayar, L. Mikeev, and V. Wolf. On the numerical analysis of stochastic Lotka-Volterra models. In *Computer Science and Information Technology (IMCSIT), Proceedings of the 2010 International Multiconference on*, pages 289–296. IEEE, 2010.
- [46] T. Dayar and M. C. Orhan. Kronecker-based infinite level-dependent QBD processes. *Journal of Applied Probability*, 49:1166–1187, 2012.
- [47] T. Dayar and M. C. Orhan. Cartesian product partitioning of multi-dimensional reachable state spaces. Technical report, BU-CE-1303, Department of Computer Engineering, Bilkent University, Ankara, Turkey, December 2013. Available online from www.cs.bilkent.edu.tr/tech-reports/2013/BU-CE-1303.pdf.
- [48] T. Dayar and M. C. Orhan. Cartesian Product Partitioning software. <http://www.cs.bilkent.edu.tr/tugrul/software.html>, 2013.

- [49] T. Dayar and M. C. Orhan. On vector-Kronecker product multiplication with rectangular factors. *SIAM Journal on Scientific Computing*, 37:S526–S543, 2015.
- [50] T. Dayar and M. C. Orhan. Vector–Kronecker Product Multiplication software. <http://www.cs.bilkent.edu.tr/tugrul/software.html>, 2015.
- [51] T. Dayar and M. C. Orhan. Cartesian product partitioning of multi-dimensional reachable state spaces. *Probability in the Engineering and Informational Sciences*, 30:413–430, 2016.
- [52] T. Dayar and M. C. Orhan. RetrialQueueSolver. <http://www.cs.bilkent.edu.tr/tugrul/software.html>, 2016.
- [53] T. Dayar and M. C. Orhan. Steady-state analysis of a multi-class MAP/PH/c queue with acyclic PH retrials. *Journal of Applied Probability*, 53:1098–1110, 2016.
- [54] T. Dayar and M. C. Orhan. Compact CME solver. <http://www.cs.bilkent.edu.tr/tugrul/software.html>, 2017.
- [55] T. Dayar, W. Sandmann, D. Spieler, and V. Wolf. Infinite level-dependent QBD processes and matrix-analytic solutions for stochastic chemical kinetics. *Advances in Applied Probability*, 43:1005–1026, 2011.
- [56] P. Deuffhard, W. Huisinga, T. Jahnke, and M. Wulkow. Adaptive discrete Galerkin methods applied to the chemical master equation. *SIAM Journal on Scientific Computing*, 30(6):2990–3011, 2008.
- [57] J. E. Diamond and A. S. Alfa. The MAP/PH/1 retrial queue. *Stochastic Models*, 14:1151–1177, 1998.
- [58] J. E. Diamond and A. S. Alfa. Approximation method for M/PH/1 retrial queues with phase type inter-retrial times. *European Journal of Operational Research*, 113(3):620–631, 1999.
- [59] V. J. Dielissen and A. Kaldewaij. Rectangular partition is polynomial in two dimensions but NP-complete in three. *Information Processing Letters*, 38(1):1–6, 1991.

- [60] K. N. Dinh and R. B. Sidje. Understanding the finite state projection and related methods for solving the chemical master equation. *Physical Biology*, 13(3):035003, 2016.
- [61] S. V. Dolgov and B. Khoromskij. Simultaneous state-time approximation of the chemical master equation using tensor product formats. *Numerical Linear Algebra with Applications*, 22:197–219, 2015.
- [62] S. V. Dolgov and D. V. Savosyanov. Alternating minimal energy methods for linear systems in higher dimensions. *SIAM Journal on Scientific Computing*, 36(5):A2248–A2271, 2014.
- [63] A. Dudin and V. Klimenok. A retrial BMAP/SM/1 system with linear repeated requests. *Queueing Systems*, 34:47–66, 2000.
- [64] S. Engblom. Spectral approximation of solutions to the chemical master equation. *Journal of Computational and Applied Mathematics*, 229(1):208–221, 2009.
- [65] G. I. Falin. On a multiclass batch arrival retrial queue. *Advances in Applied Probability*, 20:483–487, 1988.
- [66] G. I. Falin and J. G. C. Templeton. *Retrial queues*. Chapman and Hall, London, 1997.
- [67] G. Fayolle, V. A. Malyshev, and M. V. Menshikov. *Topics in the Constructive Theory of Countable Markov Chains*. Cambridge University Press, Cambridge, 1995.
- [68] P. Fernandes, B. Plateau, and W. J. Stewart. Efficient descriptor-vector multiplications in stochastic automata networks. *Journal of the ACM*, 45(3):381–414, 1998.
- [69] L. Ferrari, P. V. Sankar, and J. Sklansky. Minimal rectangular partitions of digitized blobs. *Computer Vision, Graphics and Image Processing*, 28(1):58–71, 1984.
- [70] C. Gadgil, C. H. Lee, and H. G. Othmer. A stochastic analysis of first-order networks. *Bulletin of Mathematical Biology*, 67:901–946, 2005.

- [71] T. S. Gardner, C. R. Cantor, and J. J. Collins. Construction of a genetic toggle switch in *Escherichia coli*. *Nature*, 403(6767):339–342, 2000.
- [72] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, 1979.
- [73] N. Gharbi, C. Dutheillet, and M. Ioualalen. Colored stochastic Petri nets for modelling and analysis of multiclass retrial systems. *Mathematical and Computer Modelling*, 49:1436–1448, 2009.
- [74] D. T. Gillespie. A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. *The Journal of Chemical Physics*, 22(4):403–434, 1976.
- [75] D. T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *The Journal of Chemical Physics*, 81(25):2340–2361, 1977.
- [76] D. T. Gillespie. Approximate accelerated stochastic simulation of chemically reacting systems. *The Journal of Chemical Physics*, 115(4):1716–1716, 2001.
- [77] G. H. Golub and C. F. Van Loan. *Matrix Computations*. 4th edition, Johns Hopkins University Press, Baltimore, MD, USA, 2012.
- [78] A. Gómez-Corral. A bibliographical guide to the analysis of retrial queues through matrix analytic techniques. *Annals of Operations Research*, 141:163–191, 2006.
- [79] J. Goutsias. Quasiequilibrium approximation of fast reaction kinetics in stochastic biochemical systems. *The Journal of Chemical Physics*, 122(18):184102, 2005.
- [80] J. Goutsias and G. Jenkinson. Markovian dynamics on complex reaction networks. *Physics Reports*, 529(2):199–264, 2013.
- [81] L. Grasedyck. Hierarchical singular value decomposition of tensors. *SIAM Journal on Matrix Analysis and Applications*, 31(4):2029–2054, 2010.

- [82] L. Grasedyck, D. Kressner, and C. Tobler. A literature survey of low-rank tensor approximation techniques. *GAMM-Mitteilungen*, 36(1):53–78, 2013.
- [83] W. Hackbusch. *Tensor Spaces and Numerical Tensor Calculus*. Springer, Heidelberg, Germany, 2012.
- [84] W. Hackbusch and S. Kühn. A new scheme for the Tensor representation. *Journal of Fourier Analysis and Applications*, 15(5):706–722, 2009.
- [85] R. A. Harshman. Foundations of the PARAFAC procedure: Models and conditions for an “explanatory” multi-modal factor analysis. *UCLA Working Papers in Phonetics*, 16:1–84, 1970.
- [86] Q. He, H. Li, and Y. Q. Zhao. Ergodicity of the BMAP/PH/ $s/s+K$ retrial queue with PH-retrial times. *Queueing Systems*, 35:323–347, 2000.
- [87] M. Hegland, C. Burden, L. Santoso, S. MacNamara, and H. Booth. A solver for the stochastic master equation applied to gene regulatory networks. *Journal of Computational and Applied Mathematics*, 205(2):708–724, 2007.
- [88] M. Hegland and J. Garcke. On the numerical solution of the chemical master equation with sums of rank one tensors. *ANZIAM Journal*, 52:628–643, 2011.
- [89] M. Hegland, A. Hellander, and P. Lötstedt. Sparse grids and hybrid methods for the chemical master. *BIT Numerical Mathematics*, 48:265–283, 2008.
- [90] A. Hellander and P. Lötstedt. Hybrid method for the chemical master equation. *Journal of Computational Physics*, 227(1):100–122, 2007.
- [91] W. J. Heuett and H. Qian. Grand canonical Markov model: A stochastic theory for open nonequilibrium biochemical networks. *The Journal of Chemical Physics*, 124:044110, 2006.
- [92] T. Jahnke. An adaptive wavelet method for the chemical master equation. *SIAM Journal on Scientific Computing*, 31(6):4373–4394, 2010.

- [93] T. Jahnke and W. Huisinga. Solving the chemical master equation for monomolecular reaction systems analytically. *Journal of Mathematical Biology*, 54:1–26, 2007.
- [94] T. Jahnke and W. Huisinga. A Dynamical Low-Rank Approach to the Chemical Master Equation. *Bulletin of Mathematical Biology*, 70(8):2283–2302, 2008.
- [95] T. Jahnke and T. Udrescu. Solving chemical master equations by adaptive wavelet compression. *Journal of Computational Physics*, 229(16):5724–5741, 2010.
- [96] A. Jain, S. Sahni, J. Palta, and J. Dempsey. Partitioning 3D phantoms into homogeneous cuboids. *International Journal of Foundations of Computer Science*, 14(5):905–931, 2003.
- [97] V. Kazeev, M. Khammash, M. Nip, and C. Schwab. Direct solution of the chemical master equation using quantized tensor trains. *PLoS Computational Biology*, 10, 2014.
- [98] J. M. Keil. Polygon Decomposition. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 491–518. Elsevier, Amsterdam, 2000.
- [99] B. Kim. Stability of a retrial queueing network with different classes of customers and restricted resource pooling. *Journal of Industrial and Management Optimization*, 7:753–765, 2011.
- [100] C. S. Kim, V. Mushko, and A. Dudin. Computation of the steady state distribution for multi-server retrial queues with phase type service process. *Annals of Operations Research*, 201:307–323, 2012.
- [101] J. Kim and B. Kim. A survey of retrial queueing systems. *Annals of Operations Research*, 247:1–34, 2015.
- [102] T. G. Kolda and B. W. Bader. Tensor decompositions and applications. *SIAM Review*, 51(3):455–500, 2009.

- [103] D. Kressner and F. Macedo. Low-rank tensor methods for communicating Markov processes. In G. Norman and W. Sanders, editors, *Quantitative Evaluation of Computing and Communication Systems*, Lecture Notes in Computer Science, pages 25–40. 2014.
- [104] D. Kressner and C. Tobler. **htucker** — A Matlab toolbox for tensors in hierarchical Tucker format. Technical report, August 2012. Mathematics Institute of Computational Science and Engineering, Lausanne, Switzerland, Available at <http://anchp.epfl.ch/htucker>.
- [105] D. Kressner and C. Tobler. Algorithm 941: htucker—A Matlab toolbox for tensors in hierarchical Tucker format. *ACM Transactions on Mathematical Software*, 40(3), 2014.
- [106] V. G. Kulkarni. Expected waiting times in a multiclass batch arrival retrial queue. *Journal of Applied Probability*, 23:144–154, 1986.
- [107] M. S. Kumar, K. Sohraby, and K. Kiseon. Delay Analysis of Orderly Retattempts in Retrial Queueing System with Phase Type Retrial Time. *IEEE Communications Letters*, 17:822–825, 2013.
- [108] M. Kwiatkowska, R. Mehmood, G. Norman, and D. Parker. A Symbolic Out-of-Core Solution Method for Markov Models. *Electronic Notes in Theoretical Computer Science*, 68(4):589–604, 2002.
- [109] L. De Lathauwer, B. De Moor, and J. Vandewalle. A multilinear singular value decomposition. *SIAM Journal on Matrix Analysis and Applications*, 21(4):1253–1278, 2000.
- [110] C. F. Van Loan. The ubiquitous Kronecker product. *Journal of Computational and Applied Mathematics*, 123:85–100, 2000.
- [111] A. Loinger and O. Biham. Stochastic simulations of the repressilator circuit. *Physical Review E*, 76:051917, 2007.
- [112] A. Loinger, A. Lipshtat, N. Q. Balaban, and O. Biham. Stochastic simulations of genetic switch systems. *Physical Review E*, 75:021904, 2007.

- [113] S. MacNamara, K. Burrage, and R. B. Sidje. Multiscale modeling of chemical kinetics via the master equation. *Multiscale Modeling & Simulations*, 6(4):1146–1168, 2008.
- [114] H. Maehara. Note on induced subgraphs of the unit distance graph E^n . *Discrete & Computational Geometry*, 4:15–18, 1989.
- [115] M. Mateescu, V. Wolf, F. Didier, and T. A. Henzinger. Fast adaptive uniformisation of the chemical master equation. *IET systems biology*, 4(6):441–452, 2010.
- [116] C. D. Meyer. *Matrix Analysis and Applied Linear Algebra*. SIAM, Philadelphia, PA, 2000.
- [117] D. Mitra and I. Mitrani. Analysis of a Kanban Discipline for Cell Coordination in Production Lines, II: Stochastic Demands. *Operations Research*, 5:807–823, 1991.
- [118] B. Munsky and M. Khammash. The finite state projection algorithm for the solution of the chemical master equation. *The Journal of Chemical Physics*, 124:044104, 2006.
- [119] B. Munsky and M. Khammash. A multiple time interval finite state projection algorithm for the solution to the chemical master equation. *Journal of Computational Physics*, 226:818–835, 2007.
- [120] M. F. Neuts. A versatile Markovian point process. *Journal of Applied Probability*, 16:764–779, 1979.
- [121] M. F. Neuts. *Matrix-Geometric Solutions in Stochastic Models: An Algorithmic Approach*. The Johns Hopkins University Press, Baltimore, MD, 1981.
- [122] M. F. Neuts and B. M. Rao. Numerical investigation of a multiserver retrial model. *Queueing Systems*, 7:169–190, 1990.
- [123] I. V. Oseledets. Tensor-train decomposition. *SIAM Journal on Scientific Computing*, 33(5):2295–2317, 2011.

- [124] I. V. Oseledets and S. V. Dolgov. Solution of linear systems and matrix inversion in the TT-format. *SIAM Journal on Scientific Computing*, 34(5):A2718–A2739, 2012.
- [125] R. Paige and R. E. Tarjan. Three partition refinement algorithms. *SIAM Journal on Computing*, 16(6):973–989, 1987.
- [126] T. Phung-Duc and K. Kawanishi. Multiserver retrial queues with after-call work. *Numerical Algebra, Control and Optimization*, 1:639–656, 2011.
- [127] T. Phung-Duc and K. Kawanishi. Performance analysis of call centers with abandonment, retrial and after-call work. *Performance Evaluation*, 80:43–62, 2014.
- [128] T. Phung-Duc, H. Masuyama, S. Kasahara, and Y. Takahashi. A simple algorithm for the rate matrices of level-dependent QBD processes. In *Proceedings of the 5th International Conference on Queueing Theory and Network Applications*, volume 356 of *QTNA '10*, pages 4–52. ACM, New York, NY, USA, 2010.
- [129] B. Plateau. On the Stochastic Structure of Parallelism and Synchronization Models for Distributed Algorithms. *SIGMETRICS Performance Evaluation Review*, 13(2):147–154, 1985.
- [130] B. Plateau and J.-M. Fourneau. A methodology for solving Markov models of parallel systems. *Journal of Parallel and Distributed Computing*, 12(4):370–387, 1991.
- [131] V. Ramaswami and D. M. Lucantoni. Algorithm for the multi-server queue with phase-type service. *Stochastic Models*, 1:393–417, 1985.
- [132] C. V. Rao and A. P. Arkin. Stochastic chemical kinetics and the quasi-steady-state assumption: Application to the Gillespie algorithm. *The Journal of Chemical Physics*, 118(11):4999–5010, 2003.
- [133] M. Rathinam, L. R. Petzold, Y. Cao, and D. T. Gillespie. Stiffness in stochastic chemically reacting systems: The implicit tau-leaping method. *The Journal of Chemical Physics*, 119(24):12784–12794, 2003.

- [134] H. Sakurai and T. Phung-Duc. Two-way communication retrial queues with multiple types of outgoing calls. *TOP*, 23:466–492, 2015.
- [135] W. Sandmann and V. Wolf. Computational probability for systems biology. In J. Fisher, editor, *Formal Methods in Systems Biology: First International Workshop, FMSB 2008, Cambridge, UK, June 4-5, 2008. Proceedings*, volume 5054, pages 33–47. Springer Berlin Heidelberg, 2008.
- [136] K. R. Sanft, S. Wu, M. Roh, J. Fu, R. K. Lim, and L. R. Petzold. StochKit2: software for discrete stochastic simulation of biochemical systems with events. *Bioinformatics*, 27(17):2457–2458, 2011.
- [137] L. F. Shampine and M. W. Reichelt. The MATLAB ODE Suite. *SIAM Journal on Scientific Computing*, 18(1):1–22, 1997.
- [138] M. A. Shea and G. K. Ackers. The OR control system of bacteriophage lambda: A physical-chemical model for gene regulation. *Journal of Molecular Biology*, 181(2):211–230, 1985.
- [139] Y. W. Shin. Algorithmic solution for M/M/c retrial queue with PH2 retrial times. *Journal of Applied Mathematics & Informatics*, 29:803–811, 2011.
- [140] Y. W. Shin and D. H. Moon. Approximation of M/M/c retrial queue with PH-retrial times. *European Journal of Operational Research*, 213:205–209, 2011.
- [141] Y. W. Shin and D. H. Moon. M/M/c retrial queue with multiclass of customers. *Methodology and Computing in Applied Probability*, 16:931–949, 2014.
- [142] P. L. Sjöberg, P. Lötstedt, and J. Elf. Fokker-Planck approximation of the master equation in molecular biology. *Computing and Visualization in Science*, 12:37–50.
- [143] V. Soltan and A. Gorpinevich. Minimum dissection of a rectilinear polygon with arbitrary holes into rectangles. *Discrete & Computational Geometry*, 9:57–80, 1993.

- [144] W. J. Stewart. *Introduction to the Numerical Solution of Markov Chains*. Princeton University Press, Princeton, NJ, 1994.
- [145] V. Sunkara and M. Hegland. An optimal Finite State Projection Method. *Procedia Computer Science*, 1(1):1579–1586, 2010.
- [146] M. Thattai and A. van Oudenaarden. Intrinsic noise in gene regulatory networks. *Proceedings of the National Academy of Sciences*, 98(15):8614–8619, 2001.
- [147] L. Tucker. Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31(3):279–311, 1966.
- [148] R. L. Tweedie. Sufficient conditions for regularity, recurrence and ergodicity of Markov processes. *Mathematical Proceedings of the Cambridge Philosophical Society*, 78:125–136, 1975.
- [149] E. Uysal and T. Dayar. Iterative methods based on splittings for stochastic automata networks. *European Journal of Operational Research*, 110:166–186, 1998.
- [150] V. Wolf, R. Goel, M. Mateescu, and T. A. Henzinger. Solving the chemical master equation using sliding windows. *BMC Systems Biology*, 4:42, 2010.
- [151] C. M. Woodside and Y. Li. Performance Petri net analysis of communications protocol software by delay-equivalent aggregation. In *Proceedings of the Fourth International Workshop on Petri Nets and Performance Models PNPM91*, pages 64–73, 1991.
- [152] X. Zhang, K. De Cock, M. F. Bugallo, and P. M. Djurić. A general method for the computation of probabilities in systems of first order chemical reactions. *The Journal of Chemical Physics*, 122:104101, 2005.
- [153] G. M. Ziegler. *Lectures on Polytopes*, volume 152. Springer, New York, 2012.