On Approximating the Stochastic Behaviour of Markovian Process Algebra Models

Dimitrios Milios

Doctor of Philosophy
Laboratory for Foundations of Computer Science
School of Informatics
University of Edinburgh
2014

Abstract

Markov chains offer a rigorous mathematical framework to describe systems that exhibit stochastic behaviour, as they are supported by a plethora of methodologies to analyse their properties. Stochastic process algebras are high-level formalisms, where systems are represented as collections of interacting components. This compositional approach to modelling allows us to describe complex Markov chains using a compact high-level specification.

There is an increasing need to investigate the properties of complex systems, not only in the field of computer science, but also in computational biology. To explore the stochastic properties of large Markov chains is a demanding task in terms of computational resources. Approximating the stochastic properties can be an effective way to deal with the complexity of large models. In this thesis, we investigate methodologies to approximate the stochastic behaviour of Markovian process algebra models. The discussion revolves around two main topics: approximate state-space aggregation and stochastic simulation. Although these topics are different in nature, they are both motivated by the need to efficiently handle complex systems.

Approximate Markov chain aggregation constitutes the formulation of a smaller Markov chain that approximates the behaviour of the original model. The principal hypothesis is that states that can be characterised as equivalent can be adequately represented as a single state. We discuss different notions of approximate state equivalence, and how each of these can be used as a criterion to partition the state-space accordingly. Nevertheless, approximate aggregation methods typically require an explicit representation of the transition matrix, a fact that renders them impractical for large models. We propose a compositional approach to aggregation, as a means to efficiently approximate complex Markov models that are defined in a process algebra specification, PEPA in particular.

Regarding our contributions to Markov chain simulation, we propose an accelerated method that can be characterised as almost exact, in the sense that it can be arbitrarily precise. We discuss how it is possible to sample from the trajectory space rather than the transition space. This approach requires fewer random samples than a typical simulation algorithm. Most importantly, our approach does not rely on particular assumptions with respect to the model properties, in contrast to otherwise more efficient approaches.

Lay Summary

A model is a partial representation of a system created in order to better understand it. Modelling provides an important intellectual tool for understanding complex systems. For example, modelling can give us useful insights into computer systems, such as identifying critical parts or investigating the effect of design choices.

Some systems exhibit different behaviour every time that we observe them. Systems which exhibit a degree of random behaviour are called "stochastic", meaning that there are several possible outcomes for system changes or system events. It is important to create models that capture this kind of randomness in order to be able to anticipate these different outcomes.

Stochastic process algebras are a family of modelling approaches that incorporate random behaviour. Process algebras offer a compact way to model complex systems where systems are described as collections of interacting components. A process algebra model can be translated into a Markov chain, a mathematical construct that describes all the possible interactions regarding the components involved. The behaviour of the Markov chain can be mathematically analysed in numerous ways.

This work is motivated by the need to model and explore the behaviour of complex systems. Modern computer systems typically involve many users and components, as the result of the use of new technologies such as cloud computing. Such systems give rise to very large Markov chains which require significant computing resources in order to be able to analyse their behaviour. In this thesis we present methodologies to approximate the random behaviour of complex process algebra models. The discussion revolves around two main topics: *approximate aggregation* and *stochastic simulation*.

Approximate aggregation implies that we reduce the size of the model. We discuss approaches to aggregate Markov chains, and we explain how these can be applied at the component level. The reduced model is expected to be analysed with less effort, while the results obtained will approximate the true behaviour, which otherwise would remain unknown.

Stochastic simulation is the process of producing one of the many possible trajectories through a Markov chain. Producing multiple trajectories can provide insight into the behaviour of a Markov chain. Our contribution constitutes an approach that accelerates the simulation process.

Acknowledgements

I am particularly grateful to my supervisor Stephen Gilmore for his support and guidance throughout the course of my PhD. His constructive criticism has had a great impact on my work. His annoying persistence in challenging my ideas has been great motivation to polish my work and push it further.

I thank my examiners, Pete Harrison, Isi Mitrani and Charles Sutton, for their insightful comments, which greatly helped improving this thesis.

I thank Guido Sanguinetti and Jane Hillston for their valuable feedback and comments on my work. I would also like to thank Vashti Galpin in particular, who recommended the real-world example used in the case study of Chapter 6, and Ioannis Konstas who was kind enough to provide the computational resources required.

I would like to thank my colleagues in IF-3.50, especially Ohad Kammar, Alireza Pourranjbar, Chris Banks and Anastasis Georgoulas, who have been great company and made the PhD life bearable. I have been deeply affected as a researcher by their presence and ideas.

Last but not least, many thanks to my friends Giulio Caravagna, Maria-Luisa Guerriero, Ioannis Konstas, Aggeliki Vlachostergiou and especially Myrto Avgeri, not only for their company and support all these years, but also for the stimulating discussions that I had with them.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

Chapters 3 and 4 are a refined and expanded version of [72]. Chapter 5 is based on the work presented in [73].

(*Dimitrios Milios*)

To my parents, Loukas & Maria

Table of Contents

1 Introduction **1**

2 Background **5**

2.1 Markov Chains 5

2.1.1 Discrete Time 6

2.1.2 Continuous Time 7

2.1.3 Transient Behaviour 10

2.1.4 Steady-state Behaviour 13

2.1.5 Reversibility 14

2.2 Stochastic Process Algebras 15

2.2.1 PEPA 15

2.2.2 Bio-PEPA 19

2.3 State-space Aggregation 22

2.3.1 State Equivalence 22

2.3.2 Approximate Aggregation Approaches 26

2.4 Markov Chain Simulation 26

2.4.1 Exact Methods 27

2.4.2 Approximate Simulation 28

3 Approximate Markov Chain Aggregation **31**

3.1 Near-Complete Decomposability and Spectral Segmentation 33

3.1.1 Related Work on NCD Identification 34

3.1.2 Implications of Spectral Clustering on NCD 36

3.1.3 NCD-based Aggregation 39

3.2 The Quasi-lumpability Approach 41

3.2.1 A Pseudo-metric related to Quasi-Lumpability 42

3.2.2 Formulation as a Clustering Problem 42

3.2.3 Spectral Properties and Quasi-Lumpability 44

3.3 Constructing the Aggregated Model 46

3.3.1 Approximate Lumping 47

3.3.2 Stochastic Bounds 50

3.4 Experiments and Discussion 54

3.4.1 Comparison of Aggregation Approaches 54

3.4.2 Effect on Stochastic Bounds 56

3.5 Summary 59

4 Compositional Aggregation **61**

4.1 Terminology and Conventions 62

4.2 On Equivalence of PEPA Components 63

4.2.1 Strongly Equivalent Components 64

4.2.2 Modified Strong Equivalence 66

4.3 Approximate Aggregation of Components 68

4.3.1 Real Rate Matrices 68

4.3.2 Rate Matrices with Unspecified Rates 69

4.3.3 Rate Matrices with both Active and Passive Activities 70

4.4 Semantics for Aggregated PEPA models 72

4.4.1 The Kronecker Algebra Approach 72

4.4.2 A Structured Operational Semantics Characterisation 81

4.5 Experimentation on Multi-Scale Systems 88

4.5.1 Compositional vs Global Aggregation 89

4.5.2 Component Behaviour 92

4.5.3 Experimentation with the System Size 94

4.6 Summary 96

5 Stochastic Simulation via Trajectory Sampling **99**

5.1 Related Work 100

5.2 Trajectory Sampling Simulation 102

5.2.1 Random Variables: Transition vs Trajectory Point of View . . 103

5.2.2 Sampling from the State Sequence Distribution 104

5.2.3 Sampling from the Time Distribution 108

5.2.4 Implementation Issues 111

5.3 Experiments 112

5.3.1 Evaluation of Efficiency 113

5.3.2 Evaluation of Accuracy 116
5.4 Summary 118

6 Case Study on Cloud Computing 121
6.1 Cloud Services 122
6.1.1 The Heroku Case 123
6.2 Modelling Heroku Routing with PEPA 126
6.2.1 Random Routing Policy 128
6.2.2 Smart Routing Policy 131
6.3 Evaluation of Routing Policies 134
6.3.1 Experimentation with the Workload 134
6.3.2 Experimentation with the System Size 139
6.4 Summary 142

7 Conclusions 145
7.1 Contributions on Approximate Aggregation 145
7.1.1 Approximate Aggregation Methodologies 146
7.1.2 Compositional Aggregation 147
7.2 Contributions on Simulation 148
7.3 Future Work 149

A Bio-PEPA Models of Chapter 5 151
A.1 The Schlögl Model 151
A.2 The LacY Model 152
A.3 The Goldbeter's Model 155

Bibliography 157

Chapter 1

Introduction

In the field of computer science, stochastic modelling has been a traditional approach to study the behaviour of systems that involve uncertainty [61]. *Markov chains* in particular have been extensively used for performance analysis of computer systems [89, 74]. The task of performance evaluation remains vitally important as computer infrastructures continue to expand and informational systems are integral to many aspects of life. The rise of the cloud computing industry means that the structure of systems is often extremely complex. Moreover, the massive numbers of people and devices connected to the web, for example in terms of social networks or peer-to-peer networks, not only affect the scale of the system, but also the impact of design decisions.

Markovian modelling offers a rigorous framework to investigate performance issues. A system is characterised by a set of possible states and a set of transitions each of which is associated with a probability. Staring from some initial state, the system performs a random walk over the state-space which is governed by the transition probabilities. Exploring the stochastic properties of Markovian systems involves calculating how the state probabilities change over the course of time.

The manual identification of the system state-space and the corresponding transitions can easily become a tedious task. *Stochastic process algebras* [44, 48] provide a neat and compact way to formally describe dynamic systems that exhibit stochastic behaviour, as they provide a high-level description which is amenable to rigorous mathematical analysis. Compared to other modelling formalisms, such as stochastic Petri nets [76] or queueing networks [57, 89, 74], they offer a powerful compositional framework to describe complex Markov models; systems are described as collections of interacting components, which are subsequently used to generate an underlying Markov chain. A common issue however is that sometimes even an apparently simple model specification may result in an extremely large Markov chain, a problem known as *state-space explosion*.

Applications of stochastic modelling and process algebras are not limited to computer science only. During the last fifteen years, there has been increasing interest in exploring the stochastic properties of biological systems [68, 52, 56]. Biochemical models have been traditionally treated as deterministic systems, in terms of *ordinary differential equations* (ODEs). The corresponding stochastic models have been known long ago [40], but more recently the systems biology community has shown increased interest in models that incorporate randomness. This randomness has been found to adequately capture the uncertainty observed in some intracellular biochemical processes, such as DNA transcription or protein synthesis in cells. Process algebras have been recently extended for modelling biological systems [11, 59, 23]. The use of such a formal specification for biological systems provides a unifying view of the different modelling methods available, including Markov chains and ODEs. This has the benefit that the modeller can abstract away from the technicalities associated with a certain approach, and focus on the structure of the system.

The need to deal with complex Markov chains to model computer and biological systems, and the high computational complexity of estimating the state-space probabilities has been the main motivation of this research. The current thesis is concerned with methodologies to efficiently approximate the behaviour of large Markovian models which are assumed to have been derived by a process algebra specification, more specifically PEPA [48] and Bio-PEPA [23]. Throughout the thesis, the discussion revolves around two main topics: *approximate state-space aggregation* and *stochastic simulation*.

Regarding approximate Markov chain aggregation, the elementary hypothesis is that certain sets of states can be successfully represented as a single state. It should be possible to construct a smaller Markov chain that exhibits behaviour similar to the original model, assuming that the state-space is partitioned accordingly. In the literature, Markov chain partitioning has been traditionally approached as decomposition into strongly coupled parts [27, 71, 31], which can be an appropriate criterion for state similarity in many cases. It is our thesis however that a concept of behavioural similarity, in the style of lumpability, should be able to capture a wider range of state equivalences. The following three points constitute our main contributions with respect to Markov chain aggregation: the exploration of state similarity concepts, the development of partitioning strategies, and finally the construction of an aggregated model.

The Markov chain partitioning approaches that we discuss later in this thesis intend to discover patterns in the model structure that reveal state similarities. For that purpose, we employ unsupervised machine learning algorithms, which are intrinsically computationally expensive and therefore often impractical to apply. In order to deal with the complexity of these approaches, we propose a compositional approach to aggregation, which is strongly connected with the compositional model representation that is inherent in a process algebra specification. By exploiting the component structure of PEPA models, it is possible to obtain significant state-space reduction at minimal computational overhead.

Simulation as a means of approximating stochastic properties of Markov chains is the second major topic of this work. A simulation algorithm does not require an explicit representation of the state-space. Instead, it performs a random walk and produces trajectories, which are then used to estimate the state-space probabilities or other measures. One very desirable aspect of stochastic simulation, being a Monte Carlo method, is that it converges to the true solution. That means the more sample trajectories are generated, the more accurate any estimate will be.

However, simulation can still be computationally expensive, as many simulation runs are typically required to obtain an accurate estimation of the transient behaviour of a system. In recent years, there has been increased research activity in improving the efficiency of Markov chain simulation [39, 41, 10, 7]. One particularly effective strategy to accelerate Markov chain simulation is to skip some of the simulation events [41, 10, 1]. Simulation algorithms of this type have been characterised as approximate, since they do not produce exact realisations of the Markov chain in question. The effectiveness and accuracy of approximate methods depends on whether the model complies with certain assumptions with respect to its structure. Our objective is to propose an accelerated simulation algorithm whose applicability is not dependent on the properties of the model. Instead of skipping simulation events, we focus on producing them more efficiently by reducing the amount of random numbers generated.

The topics investigated in this work are essentially of different nature, although both are motivated by the need to explore the stochastic properties of large Markov models. Compositional approximate aggregation is a method for reducing the size of the model at the cost of loss of accuracy. The approximation of the stochastic properties of Markovian systems comes in two steps: model reduction first, and solution for the state-space probabilities second. The second step may involve any of the known methods for steady-state or transient measures. In contrast, simulation directly approximates this stochastic behaviour. Minimising random number generation is a step to make the simulation process more efficient, which can be used in combination with other exact accelerated methods.

At this point, we have to emphasise that we are interested in approximating the complete stochastic behaviour of a Markov chain. In other words, our objective is to efficiently obtain approximations for the state probabilities, as opposed to other approaches such as fluid flow approximation with ODEs [49]. As a matter of fact, fluid flow approximation offers a very efficient way to have a deep insight into the system's behaviour not only with respect to the stationary measures, bur also the transient ones. However, by approximating a Markov chain with a system of ODEs, what we get is an approximation of how the mean value of several measures changes over time. Of course, such a system is inherently deterministic, implying that there is no information about stochastic properties other than the mean value [82]. Information about higher-order moments can be extracted by higher-order fluid flow methods [47], which are considerably more expensive. A deeper exploration of the stochastic behaviour requires a more traditional approach to Markov chain analysis, which can be either solving for the state-space probabilities or performing simulation.

Thesis Outline In the chapter that follows we discuss the background material and state of the art regarding state-space aggregation and stochastic simulation. The focus of Chapter 3 is the problem of approximate aggregation of unstructured Markov chains. In Chapter 4 we discuss how the structure of PEPA models can be exploited so as to efficiently apply approximate aggregation in a compositional manner. In Chapter 5 we present our accelerated stochastic simulation approach, which relies on reducing the amount of random numbers generated. In Chapter 6 we present a case study on cloud computing. More specifically, we make use of the methodologies proposed in this thesis to explore scalability issues for different routing policies in cloud services. Finally, Chapter 7 contains the concluding remarks of this work.

Chapter 2

Background

In this chapter, we discuss the background material and we establish most of the concepts and the conventions used later in the thesis. The first section is a brief introduction to the fundamental concepts of Markov chains. Section 2.2 introduces stochastic process algebras, in particular PEPA and Bio-PEPA, which are the modelling formalisms used throughout the thesis. The last two sections introduce the two approaches considered in this work to approximate the stochastic behaviour of Markovian process algebras. The first is state-space aggregation discussed in Section 2.3, which is a means to reduce the complexity of large Markov chains. The second is stochastic simulation discussed in Section 2.4, which can produce estimates for the state probabilities with no explicit representation of the Markov chain state-space. References to the related literature are given, while the works that are closely related to this thesis are discussed in more detail in the corresponding chapters.

2.1 Markov Chains

A *stochastic process* $\{X_t\}$ is a collection of random variables indexed by time. We can think of it as the stochastic analogue of a function of time. In the deterministic world, a function of time describes the evolution of some variable whose value is fixed for a given time. In the same sense, $\{X_t\}$ records the evolution of a random variable, whose value is uncertain, yet it is associated with a probability distribution over a set of possible values.

Stochastic processes are appropriate for describing the dynamic properties of systems that exhibit stochastic behaviour. Markov chains are among the most popular stochastic processes for this task, as they are supported by a plethora of techniques to obtain the probability distributions that describe their behaviour.

A Markov chain is characterised by the following two properties, which have implications which are very convenient for modelling:

- The state-space is discrete, although it does not have to be finite.
- The state probabilities are conditionally independent of the previous states given the current state. This is also known as the *Markov property*.

In order to model a system with a Markov chain, we have to define a discrete set of states and a set of probability distributions over the state-space, one for each state. Those will be the conditional state probabilities given some state, or equivalently *transition probabilities* for some state. The Markov property assures us that the transition probabilities depend solely on the current state.

This discussion actually refers to *time-homogeneous* Markov chains, for which the conditional state distribution is independent of the time that we observe some state. Time can also be part of the equation for *time-inhomogeneous* systems, although these will not be considered here. All of the Markov chains discussed in this thesis are assumed to be time-homogeneous.

2.1.1 Discrete Time

Although we are mostly interested in systems where the time index is continuous, some concepts can be more intuitively introduced for discrete-time processes. Those concepts will be later expanded to the continuous-time case.

In discrete-time systems, the notion of time is defined in terms of steps; that implies that any change of state will happen at regular intervals. Time information is recorded as number of steps n, which refers to the number of transitions that have happened since the beginning of the process. For Markov chains in a discrete-time setting we have the following definition:

Definition 1. *A finite* Discrete-Time Markov Chain *(DTMC) is a triple* $(S, P, \boldsymbol{\pi}^{(0)})$*, where* S *is a finite set of states,* P *is a* $|S| \times |S|$ *stochastic matrix, and* $\boldsymbol{\pi}^{(0)}$ *is an initial probability distribution vector over* S*.*

The P matrix was said to be stochastic, meaning that its row entries are non-negative and sum up to 1. The rows of P are actually the conditional state distributions that correspond to the set of states S. Therefore, the entry P_{ij} defines the transition probability from state i to j, where $i, j \in S$. The P matrix will be referred to as the *transition probability matrix*.

We have defined the set of states S to be finite, but it does not have to be so in the general case. The DTMC definition can be easily expanded to non-finite systems by considering a countable (and not necessarily finite) state-space and a function that maps a pair of states to some transition probability. A finite state-space however is associated with a finite transition probability matrix. Some of the results of this thesis rely on the properties of finite stochastic matrices and unless stated otherwise, the state-space S of some Markov chain shall be considered to be finite.

Alternatively, we can define a DTMC as a discrete state stochastic process $\{X_n\}$ that takes values on S with initial probabilities $X_0 = \boldsymbol{\pi}^{(0)}$. In order to be a Markov process, $\{X_n\}$ has to satisfy the following condition:

$$Pr(X_{n+1} = s_{n+1} \mid X_0 = s_0, \ldots X_n = s_n) = Pr(X_{n+1} = s_{n+1} \mid X_n = s_n) \tag{2.1}$$

where $n \in \mathbb{N}_0$ is the time index, and $s_n \in S$ denotes the state at time n. Equation (2.1) formally describes the Markov property for DTMCs. Time corresponds to the number of steps or transitions which have occurred. Therefore, the random variable X_n corresponds to the state probability distribution after n transitions.

Given some state-space and the corresponding transition probabilities, we can perform a random walk, starting from some initial distribution $\boldsymbol{\pi}^{(0)}$. In order to describe all the possible paths in the random walk, we have to calculate the unconditional state probabilities at different times. Our objective is to estimate the distribution of the X_n random variables at different times n, when we know the conditional probabilities $Pr(X_{n+1} \mid X_n)$. Recall that for a time-homogeneous process, these conditional probabilities do not depend on time, hence we have: $Pr(X_{n+1} = j \mid X_n = i) = P_{ij}$.

2.1.2 Continuous Time

In a DTMC, the time of the next transition is actually deterministic, as we have exactly one transition per time-step. In many occasions we want to model systems where discrete changes of state may happen at random times that can be any non-negative real number. Each transition is associated with a non-negative real-valued random variable that represents its duration.

2.1.2.1 Implications of the Markov Property

The duration of each transition in a continuous time setting is known to be exponentially distributed. As we shall see next, this is a direct consequence of the Markov property. Let us consider a discrete state stochastic process $\{X_t\}$ that takes values on S, where $t \geq 0$ is a continuous-time index. Then $\{X_t\}$ will be a *Continuous-Time Markov Chain* (CTMC) if it satisfies the Markov property, which is captured by the following equation for any $h \geq 0$:

$$Pr(X_{t+h} = j \mid X_t = i, \{X_\tau : 0 \leq \tau \leq t\}) = Pr(X_{t+h} = j \mid X_t = i) \tag{2.2}$$

According to Equation (2.2), the probability of moving from state $i \in S$ to state $j \in S$ after time h, is independent of any previously visited states. In the continuous-time case, there is no notion of next time-step, so the Markov property has to be defined for an arbitrary time interval h.

However, there is the notion of next transition. Recall that the state-space is discrete, so the value of $\{X_t\}$ remains constant between the transition times. If we have k transitions in $\{X_t\}$, then there is a sequence $0 < t_1 < t_2 \cdots < t_k$ that represents the times when these transitions occur. Because the Markov property as expressed in Equation (2.2) holds for arbitrary h, for the transition at time t_{k+1} we have:

$$Pr(X_{t_{k+1}} = s_{k+1} \mid X_{t_1} = s_1, \ldots X_{t_k} = s_k) = Pr(X_{t_{k+1}} = s_{k+1} \mid X_{t_k} = s_k) \tag{2.3}$$

where $s_k \in S$ denotes the state after k transitions. Equation (2.3) states that the next step probabilities at time t_{k+1} depend solely on the state at t_k. If we ignore the transition times what we have is a discrete-time process, which is essentially a DTMC. This is called the *jump process* or *embedded Markov chain* that defines the transition probabilities for a CTMC. Since the CTMC considered is time-homogeneous, so will the corresponding jump process be, meaning that the next step probabilities do not depend on the time t_k.

The jump process however does not fully describe the behaviour of some CTMC. We have to somehow model the fact that the transitions occur at random moments. Consider the sequence of continuous random variables $T_1, T_2, \ldots, T_k$ that denote the times at which k transitions happen. The sequence of random variables $L_1, L_2, \ldots, L_k$ will represent the times between transitions, i.e. $L_1 = T_1, L_2 = T_2 - T_1$ and so forth. These times are called *holding* or *sojourn times*. The distribution of these random variables has to be such that the Markov property is satisfied.

At the times when the transitions occur, we have seen that the Markov property holds if we consider a discrete-time jump process that is a DTMC. What about the time intervals in between the transitions? Given some state i at time t, let us consider the probability that no transition happens within an interval $u+v$, where $u > 0$ and $v > 0$. We then break this probability down by applying Bayes rule:

$$\begin{aligned} Pr(L > u+v \mid X_t = i) &= Pr(L > u, L > u+v \mid X_t = i) \\ &= Pr(L > u+v \mid L > u, X_t = i) \cdot Pr(L > u \mid X_t = i) \end{aligned} \tag{2.4}$$

If we consider a time-homogeneous system, we can set $u = 0$ for the first term of the product above. In this way, the condition $L > 0$ can be ignored, as it is one the initial assumptions. Then we have:

$$Pr(L > u+v \mid X_t = i) = Pr(L > v \mid X_t = i) \cdot Pr(L > u \mid X_t = i) \tag{2.5}$$

Equation (2.5) implies that the distribution of the L random variable has to be memoryless. The only continuous memoryless distribution is the exponential distribution. Indeed, considering $L \sim Exp(\lambda_i)$ with $P(L \leq t) = 1 - e^{-\lambda_i t}$ or $P(L > t) = e^{-\lambda_i t}$, it is easy to verify that $e^{-\lambda_i (u+v)} = e^{-\lambda_i u} e^{-\lambda_i v}$. The rate λ_i of the exponential distribution that governs the sojourn time depends only on the current state i, as can be seen in Equation (2.5).

2.1.2.2 Continuous-Time Markov Chains

To summarise, a CTMC is fully characterised in terms of a jump process and a collection of exponentially distributed holding times. Since we are interested in time-homogeneous systems only, the holding times depend solely on the state. Below we present an alternative and more convenient definition for CTMCs that is equivalent.

Definition 2. *A Continuous-Time Markov Chain (CTMC) is a triple $(S, Q, \boldsymbol{\pi}_0)$, where S is a finite set of states, Q is a $|S| \times |S|$ generator matrix, and $\boldsymbol{\pi}_0$ is the initial probability distribution over S.*

A matrix Q is called a *generator matrix* if its entries Q_{ij} for any $i, j \in S$ satisfy the following properties :

(i) $Q_{ij} \geq 0, \quad i \neq j$

(ii) $Q_{ii} = -\sum_{j \in S, i \neq j} Q_{ij}$

A non-diagonal entry Q_{ij} denotes the *transition rate* from state i to state j. This rate can be interpreted as the parameter of an exponential random variable $L_{ij} \sim Exp(Q_{ij})$ that determines the time of transitioning from i to j. In this representation, each state is associated with several exponential random variables.

When two or more transitions are possible, then there is a *race condition* between them. In such a case, the transition that will trigger first will be the only one to update the system state. Therefore the sojourn time is the minimum of $|S|-1$ exponential random variables $L_i = \min(\{L_{ij} : j \in S, j \neq i\})$, which is known to be also exponentially distributed with parameter $Q_i = \sum_{j \in S, i \neq j} Q_{ij}$.

The jump process is DTMC with probability matrix P with entries:

$$P_{ij} = \begin{cases} Q_{ij}/Q_i, & i \neq j \text{ and } Q_i \neq 0 \\ 0, & \text{otherwise} \end{cases}, \quad \text{where } Q_i = \sum_{j \neq i} Q_{ij} \tag{2.6}$$

Hence, the next transition probabilities given some state i are encoded in the i-th row P, while the time of this next transition follows exponential distribution with rate Q_i.

2.1.3 Transient Behaviour

In this section we briefly discuss how the stochastic behaviour of Markov chains is calculated. We have described Markov chains in terms of their transition probabilities in the discrete-time case, or their transition rates for continuous-time systems. We want to calculate the distributions of the random variables that constitute some Markovian stochastic process $\{X_t\}$. We focus on methods that calculate the exact state distributions, as their high computational complexity motivated the development of approximation approaches.

The state distribution of a Markov chain at different times is referred to as *transient behaviour*, as the distribution over the state-space changes as time proceeds. We shall introduce some concepts for the discrete-time case, before we deal with CTMCs.

2.1.3.1 Transient Probabilities for DTMCs

Let $\{X_n\} \equiv (S, P, \boldsymbol{\pi}^{(0)})$ be some finite DTMC. We know that X_0 follows a categorical distribution with parameters contained in the probability vector $\boldsymbol{\pi}^{(0)}$. In a finite-state system, the change on the state distribution can be written in terms of the transition probability matrix using linear algebra. According to the definition of DTMCs, the next random variable X_1 is also categorical with parameters contained in $\boldsymbol{\pi}^{(1)} = \boldsymbol{\pi}^{(0)} P$.

In the general case, the n-th step probabilities for $\{X_n\}$ are given by the n-th power of P as follows:

$$\boldsymbol{\pi}^{(n)} = \boldsymbol{\pi}^{(0)} P^n \tag{2.7}$$

Equation (2.7) offers a way to calculate exactly the probabilities at any stage of the lifetime of the stochastic process. The disadvantage however is that the calculation involves raising a matrix to a power, which is a problem of high complexity, unless the matrix has some special structure (i.e. diagonalisable).

Instead, we can take advantage of the Markov property and time-homogeneity. That is, given some current state, the transition probabilities are independent of the time this state is observed. So for any state distribution $\boldsymbol{\pi}^{(n)}$, the next step probabilities are given as following:

$$\boldsymbol{\pi}^{(n+1)} = \boldsymbol{\pi}^{(n)} P \tag{2.8}$$

In this way, we can recursively calculate the state probabilities at any time n, starting from the initial distribution vector.

2.1.3.2 Transient Probabilities for CTMCs

In order to calculate the transient state probabilities in the continuous-time case, we need to express the transition probabilities in terms of time. According to [79], the transition probabilities of a CTMC after time t are given by the following function:

$$P(t) = e^{tQ} \tag{2.9}$$

where Q is the corresponding generator matrix. In fact, $P(t)$ is the continuous analogue of the P^n matrix, which denotes the transition probabilities after n steps in a DTMC. Given an initial state distribution vector $\boldsymbol{\pi}_0$, the distribution vector of the CTMC at time t will be:

$$\boldsymbol{\pi}_t = \boldsymbol{\pi}_0 P(t) \tag{2.10}$$

$P(t)$ can be calculated as a weighted sum of different powers of the probability matrix P of the underlying jump process. The state distribution at t can then be rewritten as follows:

$$\boldsymbol{\pi}_t = \boldsymbol{\pi}_0 \sum_{k=0}^{\infty} P^k \times Pr(k \text{ steps until } t) \tag{2.11}$$

The equation above is problematic though, as it involves an infinite sum which, itself contains several powers of P. Moreover, the probability of making k transitions until time t is in general difficult to estimate.

One way to reduce the cost of (2.11) is to employ the technique of *uniformisation* [54]. If $\{X_t\} \equiv (S, Q, \boldsymbol{\pi}^{(0)})$ is some finite CTMC, let λ be such that:

$$\lambda = \sup_{i \in S} \sum_{j \neq i} Q_{ij} \tag{2.12}$$

This λ will be called *uniformisation rate*. We can then construct a DTMC with the following probability matrix:

$$P' = I + \frac{1}{\lambda} Q \tag{2.13}$$

This will be an embedded Markov chain that is however different from P as defined in Equation (2.6). Recall that the duration of k transitions is determined by a collection of exponential random variables $L_1, L_2, \ldots, L_k$, the rate of each of which depends on the current state. Uniformisation modifies the jump process to P' in such a way that the random variables $L'_1, L'_2, \ldots, L'_k$ follow an exponential distribution with the same rate λ. This has the consequence that the number of jumps at different times is governed by a Poisson process. Therefore the number k of transitions until time t follows a Poisson distribution with parameter λt. So we have:

$$Pr(k \text{ steps until } t) = e^{-\lambda t} \frac{(\lambda t)^k}{k!} \tag{2.14}$$

We can rewrite Equation (2.11) using P' as the jump process and Equation (2.14) to calculate the step probabilities. We can also replace $\boldsymbol{\pi}_0 P'^k$ with $\boldsymbol{\pi}^{(k)}$:

$$\boldsymbol{\pi}_t = \boldsymbol{\pi}_0 \sum_{k=0}^{\infty} P'^k \times e^{-\lambda t} \frac{(\lambda t)^k}{k!} = \sum_{k=0}^{\infty} \boldsymbol{\pi}^{(k)} \times e^{-\lambda t} \frac{(\lambda t)^k}{k!} \tag{2.15}$$

The vectors $\boldsymbol{\pi}^{(k)}$ represent the state probabilities at different stages of the embedded Markov chain that is produced after uniformisation. To estimate the state probabilities of a CTMC at any time t we have to calculate $\boldsymbol{\pi}^{(k)}$ for several values of k. These can be calculated recursively as done in the discrete-time case, and for sparse matrices $\boldsymbol{\pi}^{(k)}$ can be calculated efficiently.

Note that the infinite sum is still present in Equation (2.15). The approach is only exact if we consider an infinite number of jumps. Normally we truncate the summation to K terms that will result in an approximate probability vector $\tilde{\boldsymbol{\pi}}_t$, however the method can be made arbitrarily precise by choosing a K large enough. In this thesis, uniformisation is used as a baseline to evaluate the accuracy with respect to the transient behaviour of some of the approaches proposed.

2.1.4 Steady-state Behaviour

The long-term properties of Markov chains are of particular interest. If certain conditions hold, the state-space probabilities in the long run converge to a unique *steady-state* distribution. In the case of DTMCs, a stationary probability vector $\boldsymbol{\pi}$ satisfies the following property:

$$\boldsymbol{\pi} P = \boldsymbol{\pi} \tag{2.16}$$

This means that if the state distribution at any step is $\boldsymbol{\pi}$, this will never alter. The existence of one or more stationary distributions depends on the properties of the Markov chain.

Irreducibility We say that a state j can be reached from a state i, if there non-zero probability of moving from i to j in a random walk. A state i is said to communicate with j, if j can be reached from i and vice versa. The state-space S of a Markov chain can be partitioned into *communicating classes*. A communicating class is a set of states $C \subseteq S$ such that any pair $i, j \in C$ communicates with each other, while no state $i \in C$ communicates with any state $j \notin C$. If the state-space S is a single communicating class, then the Markov chain is called *irreducible*. For an irreducible Markov chain, there exists at most one stationary distribution [55, 79, 45].

Aperiodicity Given A Markov chain $\{X_n\}$, a state i is said to be *aperiodic* if and only if the set $\{n : Pr(X_n = i \mid X_0 = i) > 0\}$ has no common divisor other than 1 [79]. Intuitively, this means that a random walk can return to state i at irregular times. A Markov chain is called aperiodic, if all of its states are aperiodic. For an aperiodic Markov chain, there exists at least one stationary distribution [55, 79, 45].

Ergodicity A Markov chain is said to be *ergodic* if it is aperiodic and irreducible [55, 79, 45]. It follows that an ergodic chain has exactly one stationary probability vector. The calculation of the steady-state probabilities for ergodic chains is well established. For DTMCs, the steady-state distribution vector is the eigenvector of the transition probability matrix that corresponds to the eigenvalue $\lambda = 1$, as we can see in Equation (2.16). For a CTMC with generator matrix Q, the stationary vector $\boldsymbol{\pi}$ satisfies the *global balance equation*:

$$\boldsymbol{\pi} Q = 0 \tag{2.17}$$

What we have is a system of equations where the $\boldsymbol{\pi}$ vector is the unknown. This system can be solved by considering the following additional condition that stems from the fact that $\boldsymbol{\pi}$ is a probability vector:

$$\sum_{i \in S} \boldsymbol{\pi}_i = 1 \tag{2.18}$$

There are many approaches to efficiently calculate the steady-state probabilities of ergodic Markov chains. In the general case however, a Markov chain might not be ergodic, meaning that it might have more than one stationary behaviour. Steady-state analysis does not produce any information with respect to what happens until some equilibrium is reached. Such information is only accessible through transient analysis, which is typically much more computationally expensive, as the solution involves several random variables. The stochastic behaviour that we seek to approximate in terms of this thesis refers to both transient and steady-state properties. The high computational complexity of the approaches to exactly calculate the state probabilities motivated the use of state-space aggregation and efficient stochastic simulation as tools to approximate the state distribution in a more efficient way.

2.1.5 Reversibility

In this section, we introduce some notions regarding time reversal of Markov chains, which will be discussed later in this thesis. An irreducible Markov chain with transition probability matrix P, state-space S, and steady-state distribution $\boldsymbol{\pi}$ is said to be *reversible* if the *detailed balance equation* holds:

$$\pi_i P_{ij} = \pi_j P_{ji}, \quad \forall i, j \in S \tag{2.19}$$

Intuitively, this means that when in steady-state, the probability of each possible transition is equal to the probability of the reversed transition. Reversibility is related to a class of approximate aggregation methodologies discussed in Chapter 3, which exploit the symmetric property of the detailed balance equation in (2.19).

For any Markov chain, it is possible to define its *time-reversal*, which is also a Markov chain with transition probability matrix $\bar{P}$, whose elements are defined as follows:

$$\bar{P}_{ij} = P_{ji} \frac{\pi_j}{\pi_i} \tag{2.20}$$

It can be easily shown that a Markov chain and its time-reversal have the same steady-state distribution [55, 79]. In the reversible case, we have $P = \bar{P}$.

2.2 Stochastic Process Algebras

Process algebras are languages that describe systems as collections of interacting entities called *agents*. Each agent may perform a number of *actions*, which can be carried out independently or in collaboration with other agents. Process algebras were originally used for modelling concurrency in systems that did not exhibit any stochastic behaviour. The stochastic extensions in process algebras are achieved by associating each action with a random variable that corresponds to its duration [44, 48]. If these random variables are chosen to be exponentially distributed, then the process algebra is essentially Markovian, meaning that it can be mapped to a CTMC. The next two subsections briefly introduce the Markovian process algebras considered in this thesis.

2.2.1 PEPA

PEPA [48] is among the first process algebras that made use of exponentially delayed actions. Its acronym stands for *Performance Evaluation Process Algebra*, which expresses the intention to capture quantitative properties of systems, including performance measures such as utilisation or throughput. PEPA models are collections of *components*; the modeller has to specify the components and the way these components interact with each other. The combination of the components can be mapped to a CTMC that can be solved for the transient and the steady-state behaviour of the system. More formally, the grammar for the PEPA language is the following:

$$\begin{array}{rcl} S & ::= & (\alpha, r).S \mid S + S \\ P & ::= & P \underset{\mathcal{L}}{\bowtie} P \mid P/\mathcal{L} \mid S \end{array}$$

where S denotes a sequential component, while P denotes a parallel component which is defined as a composition of sequential components. Below we explain the meaning of the operators and the notation used.

Prefix (.) The prefix operator describes a sequential action that a component may perform. For example, a component $(\alpha, r).P$ carries out an action and subsequently behaves as P. The pair (α, r) is called an *activity*, where α is the action type and r is the *rate* of the activity. The duration of the activity is governed by an exponential distribution with mean $1/r$. The set of activities that a component P is capable of is denoted as $\mathcal{A}ct(P)$.

Choice $(+)$ This operator denotes a choice between two different sequential behaviours. A component $P+Q$ can evolve either to P or Q. Whenever there is a choice between two or more activities, the exponentially distributed transition times imply that there is a race condition between them. For instance, the component $P \overset{def}{=} (\alpha_1, r_1).P_1 + (\alpha_2, r_2).P_2$, will either perform α_1 with probability $r_1/(r_1+r_2)$ or α_2 with probability $r_2/(r_1+r_2)$, while the time of exiting state P will follow an exponential distribution with rate r_1+r_2.

Cooperation ($\underset{\mathcal{L}}{\bowtie}$) This operator denotes a parallel composition of interacting components. A cooperation is defined over a set of actions $\mathcal{L}$, which is called the *cooperation set*. The actions in this set are also called *shared* actions, and they require that the components involved carry out the activity simultaneously. Components may carry out individually any activity whose action type is not in the cooperation set. A cooperation can also be defined on an empty set of actions, meaning that the components are independent.

Empty Cooperation $(||)$ This operator is a shorthand for $\underset{\emptyset}{\bowtie}$, where the cooperation set is empty.

Aggregation $([N])$ The notation $S[N]$ denotes a collection of identical sequential components that act in parallel with no interaction among them. Alternatively, we could write $S||S||\ldots S$ where S occurs N times.

Hiding $(/)$ Hiding $P/\mathcal{L}$ renders the actions in the set $\mathcal{L}$ private for the component P. This means that no cooperation can be defined for the actions contained in $\mathcal{L}$. The components are forced to carry out their private activities independently. As the grammar notation implies, P might be either sequential or parallel.

Unspecified Rates The rate of an activity can be *unspecified*, denoted by $\top$, meaning that the activity is *passive*. A passive activity can only be carried out in collaboration with another component, otherwise the activity is just disabled.

Although the rate of passive activities is unspecified, they can also be associated with a probability. This is required if more than one passive activities of the same action type are enabled. Then an unspecified activity rate $\top$ may be assigned a weight $w \in \mathbb{N}$ which represents the relative probability of that particular activity. The absence of a weight simply implies that $w = 1$. For example, the component $P \overset{def}{=} (\alpha_1, w_1 \times \top).P_1 + (\alpha_2, w_2 \times \top).P_2$, will perform α_1 with probability $w_1/(w_1 + w_2)$ or α_2 with probability $w_2/(w_1 + w_2)$. The time of exiting state P remains unspecified.

2.2.1.1 The Derivation Graph

The Markovian interpretation of a PEPA model relies on the semantics of the PEPA language [48] summarised in Figure 2.1, which are defined in the style of Plotkin's *structured operational semantics* [80]. For a component P, the PEPA semantics induces the set of states reachable from P, which is called the *derivative set*, denoted as $ds(P)$. The derivative set along with the activities involved define the *derivation graph*. This is actually a labelled multi-transition system with states in $ds(P)$ and transitions in $ds(P) \times \mathcal{A}ct(P) \times ds(P)$. If the derivation graph contains no unspecified rates, then we can construct a CTMC $(ds(P), Q, \boldsymbol{\pi}_0)$, where $\boldsymbol{\pi}_0$ is some initial distribution over $ds(P)$, while Q is a generator matrix whose entries capture the transition rates of the derivation graph.

In the first level of syntax, the modeller defines one or more sequential components. The second level of syntax is the *system equation*, which specifies which components participate in the system and what are the interactions among them. The local states of the components formulate the global state of the system. Any change in the local states will also have an effect on the system state. In the case of individually performed actions, the transitions are associated with an exponential distribution as described earlier. For a shared action however, the components involved have to perform this action at the same time. The duration of a synchronised transition will follow an exponential distribution that is determined by the activity with the smallest rate. In order to see how, we have to explain the notion of *apparent rate* introduced in [48]:

Definition 3 (Apparent Rate)**.** *The apparent rate of an action α in a component P, which is denoted as $r_\alpha(P)$, is the sum of all rates of all activities of type α in $\mathcal{A}ct(P)$.*

As with activity rates, an apparent rate can be either a positive real number, or unspecified $\top$ with weight equal to the sum of the weights of the activities included. Note however that the apparent rate $r_\alpha(P)$ can only be defined if the activities of type α for P are either all active or all passive, since the addition of an active rate $r \in R^+$ with an unspecified rate $\top$ is not defined as an operation in [48]. Components for which any action of type α involves both active and passive activities are considered invalid. This

Prefix

$$\frac{}{(\alpha, r).E \xrightarrow{(\alpha,r)} E}$$

Cooperation

$$\frac{E \xrightarrow{(\alpha,r)} E'}{E \underset{L}{\bowtie} F \xrightarrow{(\alpha,r)} E' \underset{L}{\bowtie} F} \ (\alpha \notin L) \qquad \frac{F \xrightarrow{(\alpha,r)} F'}{E \underset{L}{\bowtie} F \xrightarrow{(\alpha,r)} E \underset{L}{\bowtie} F'} \ (\alpha \notin L)$$

$$\frac{E \xrightarrow{(\alpha,r_1)} E' \quad F \xrightarrow{(\alpha,r_2)} F'}{E \underset{L}{\bowtie} F \xrightarrow{(\alpha,R)} E' \underset{L}{\bowtie} F'} \ (\alpha \in L) \qquad \text{where } R = \frac{r_1}{r_\alpha(E)} \frac{r_2}{r_\alpha(F)} \min(r_\alpha(E), r_\alpha(F))$$

Choice

$$\frac{E \xrightarrow{(\alpha,r)} E'}{E + F \xrightarrow{(\alpha,r)} E'} \qquad \frac{F \xrightarrow{(\alpha,r)} F'}{E + F \xrightarrow{(\alpha,r)} F'}$$

Hiding

$$\frac{E \xrightarrow{(\alpha,r)} E'}{E/L \xrightarrow{(\alpha,r)} E'/L} \ (\alpha \notin L) \qquad \frac{E \xrightarrow{(\alpha,r)} E'}{E/L \xrightarrow{(\tau,r)} E'/L} \ (\alpha \in L)$$

Constant

$$\frac{E \xrightarrow{(\alpha,r)} E'}{A \xrightarrow{(\alpha,r)} E'} \ (A \overset{def}{=} E)$$

Figure 2.1: Structured operational semantics for PEPA

requirement has some interesting implications in an approximate aggregation setting, which we explore later in Section 4.3.3.

The concept of apparent rate refers to the rate of any currently enabled activity of a given action type. The rate of a synchronised activity will be the minimum of the two apparent rates involved, weighted by the individual activity probabilities. More specifically, the rate of the synchronised activities $P \overset{def}{=} (\alpha, r_1).P'$ and $Q \overset{def}{=} (\alpha, r_2).Q'$ will be

$$r = \frac{r_1}{r_\alpha(P)} \frac{r_2}{r_\alpha(Q)} \min(r_\alpha(P), r_\alpha(Q)) \tag{2.21}$$

In the special case where there is one or more passive actions, the unspecified rate $\top$

is evaluated in terms of the minimum function as follows:

$$\min(w_1 \times \top, r) = \begin{cases} r, & r \in \mathbb{R}^+ \\ \min(w_1, w_2) \times \top, & r = w_2 \times \top \end{cases} \tag{2.22}$$

where $w_1, w_2 \in \mathbb{N}$.

The use of the minimum rate in (2.21) captures the fact that a coordinated activity is essentially determined by the slowest component. A classical approach to determine the duration of a synchronised transition based on the slowest component, would be to consider the maximum of the durations for the activities involved in the cooperation. However, the maximum of two exponentially distributed random variables does not follow an exponential distribution in the general case. In the PEPA language, maximum duration is approximated by an exponential distribution whose rate is given by (2.21). This approximation is a major assumption of the language, as defined in [48].

2.2.2 Bio-PEPA

Recently, there has been significant interest in using process algebras for modelling biological systems [11, 23]. Bio-PEPA [23] is a variation of PEPA, where processes are defined in a manner that is more convenient for the description of systems such as chemical reaction networks. In order to better understand the challenges of this kind of modelling, we have to introduce some fundamental concepts.

In the relevant literature [92], a biological system is usually described as a network of coupled chemical *reactions* that have an effect on a number of *species*. Species are the different kinds of molecular populations existing in a system. A generic form of a reaction network consisting of $N \in \mathbb{N}$ species S_n, with $1 \leq n \leq N$, and $M \in \mathbb{N}$ reaction channels R_m, with $1 \leq m \leq M$, is the following:

$$r_1^{(m)} S_1 + r_2^{(m)} S_2 + \cdots + r_N^{(m)} S_N \longrightarrow p_1^{(m)} S_1 + p_2^{(m)} S_2 + \cdots + p_N^{(m)} S_N \tag{2.23}$$

The species appearing on the left hand side of the reaction above are called *reactants*. These are accompanied by coefficients $r_n^{(m)}$ called *stoichiometries*, which denote the number of S_n molecules consumed by a single reaction R_m. The right hand side species are the *products* of the reaction, and equivalently, the stoichiometries $p_n^{(m)}$ denote the number of S_n molecules produced by a single reaction R_m. Of course, if a stoichiometry $r_n^{(m)}$ or $p_n^{(m)}$ is constantly zero, there is no need for the n-th species to be present in the definition of the m-th reaction, either as reactant or as product correspondingly.

Furthermore, each reaction has one more parameter that has not appeared in Equation (2.23). That is a rate, or more accurately, a *propensity function* $a_m(X, c_m)$, which is dependent on the current species populations, denoted by $X = \{X_1, \ldots, X_N\}$, and a rate constant c_m. These rates are actually the inverse means of exponential distributions underlying the stochastic kinetics of the system. Thus, the time that any reaction happens is governed by a exponential distribution with rate:

$$a(X, c) = \sum_{m=1}^{M} a_m(X, c_m) \tag{2.24}$$

It is noted that a common assumption in the field is that the system is well stirred and in thermal equilibrium. This means that molecules interact at a constant temperature, and are homogeneously distributed in some fixed volume. Therefore, the reaction rates depend solely on the populations and not on the relative position of the molecules. A biological system described in these terms can be mapped to a time-homogeneous CTMC whose state-space is all the different molecular populations possible for the species considered. The transitions between states are associated with exponentially distributed delays that depend on the current state. Bio-PEPA is a reagent-centric approach to modelling biological systems, where systems are described in terms of their components.

Early attempts to model biological systems using PEPA were impeded, since there is no mechanism in PEPA to describe notions such as stoichiometry or propensity functions. Components in Bio-PEPA do not involve transitions between user-defined states, in contrast with PEPA. Instead, a component corresponds to the population levels of a certain species. A Bio-PEPA model consists of a collection of species, as well as a collection of reactions that modify the species populations. The Bio-PEPA language is formally defined by the following grammar:

$$\begin{array}{rcl} S & ::= & (\alpha, \kappa)\mathtt{op}S \mid S + S \\ P & ::= & P \underset{\mathcal{L}}{\bowtie} P \mid S(N) \end{array}$$

where S denotes a sequential or species component. The way that the population of some species S may be modified by a certain reaction is specified in the term $(\alpha, \kappa)\mathtt{op}$, where α denotes the reaction type and κ represents the stoichiometry. The role of the species S in some particular reaction is described by the prefix combinator "$\mathtt{op}$" which can be any of the following:

Reactant ($\downarrow$) The reaction α will decrease the species population by the number denoted by the stoichiometry κ.

Product ($\uparrow$) The reaction α will increase the species population by the number denoted by the stoichiometry κ.

Activator ($\oplus$) The species activates the reaction α, but the species population is not affected.

Inhibitor ($\ominus$) The species inhibits the reaction α, but the species population is not affected.

Generic modifier ($\odot$) The species has some unspecified effect on the rate of the reaction α, but the species population is not affected.

The *model component* P is a parallel composition of species components by the $\underset{\mathcal{L}}{\bowtie}$ operator. The *cooperation set* $\mathcal{L}$ is a set of reaction types over which a cooperation is defined. As in PEPA, collaborating components have to perform simultaneously any action that is in the cooperation set, whereas they can perform the rest of the actions independently. The global state of the system is a vector $X = \{X_1, \ldots, X_N\}$ that contains the populations of all the species involved in the model. According to the grammar definition, a species component may participate in one or more reactions that possibly affect its population and the global state as a result. The sequential components are initialised according to the term $S(N)$, where N denotes the initial population for the species S.

As discussed in the context of chemical reaction systems, each reaction is associated with a propensity function $a_m(X, c_m)$. This is evaluated to a rate corresponding to an exponential random variable that determines the amount of time until the firing of that reaction. The propensity functions typically depend on the reactants, the modifiers (activators, inhibitors or generic modifiers) and a rate constant c_m. In the general case however, they can be arbitrary functions of the current state.

The Markovian interpretation of a Bio-PEPA model relies on the semantics of the language, which induces a labelled transition system from which a CTMC is derived. The structured operational semantics of Bio-PEPA can be found in [23].

2.3 State-space Aggregation

The Markovian process algebras discussed in the previous section can be powerful tools to describe complicated CTMCs using a minimal high-level specification. It is definitely desirable to describe complicated systems, this however will naturally increase the demand for computational resources.

State-space aggregation can be an effective way to reduce the complexity of large Markov models. Aggregated models feature a reduced number of states, a fact that can accelerate transient and steady-state analysis techniques. Aggregation can be either exact or approximate. Exact aggregation of a Markov chain involves constructing a model with a smaller number of states that exhibits behaviour identical to that of the original system. If the original model is *lumpable*, then the resulting aggregated model will be a Markov chain as well. In the case of non-lumpable models, we use a reduced Markov model that approximates the behaviour of the original system. In this way, the model can be solved efficiently at the cost of loss of accuracy.

2.3.1 State Equivalence

In order to aggregate a Markov chain with N states, its state-space has to be partitioned into K *classes*, where ideally $K \ll N$. Given a Markov chain with state-space S, we shall say that $\Delta = \{A_1, \ldots, A_K\}$ is a *partition* on S with K classes, where $A_1, \ldots, A_K$ are mutually disjoint subsets of S. More formally, for any $A_k, A_l \in \Delta$ with $k \neq l$ we have $A_k, A_l \subseteq S$, $A_k \cap A_l = \emptyset$, and $A_1 \cup A_2 \cup \ldots A_K = S$.

The states that belong to the same class have to be equivalent in some sense. In this section, we review some notions of exact and approximate state equivalence in Markov chains. A large part of the discussion in this thesis relies on the concepts that follow.

2.3.1.1 Lumpability

In terms of Markov chains, equivalence is formally described by the notion of *lumpability* [55]. Given a partition of the state-space, lumpability implies that states that belong to the same class have identical transition probabilities to each of the partitions. This concept is formally described by the following definition:

Definition 4 (Lumpability). *A Markov chain with probability matrix P is lumpable w.r.t. a partition $\Delta = \{A_1, \ldots, A_K\}$, if for any two classes $A_k, A_l \in \Delta$, and for any two* *states $i, j \in A_k$:*

$$\sum_{m \in A_l} P_{im} = \sum_{m \in A_l} P_{jm} \tag{2.25}$$

As can be seen in [6], given a lumpable Markov chain we can obtain a *lumped* model which is also a Markov chain. Given a stochastic matrix $P \in \mathbb{R}^{N \times N}$ that is lumpable to $\Delta = \{A_1, \ldots, A_K\}$, we define the corresponding lumped matrix $\tilde{P} \in \mathbb{R}^{K \times K}$ with entries:

$$\tilde{P}_{kl} = \sum_{j \in A_l} P_{ij}, \quad \forall i \in A_k \tag{2.26}$$

Lumpability dictates that the sums above are constant for all the states in the same class. Therefore, when calculating the transition probability from A_k to A_l, it does not really matter in which state of A_k the system is in. That is a direct consequence of the fact that the behaviour of the states in A_k is exactly equivalent with respect to Δ. The lumped matrix $\tilde{P}$ has transient and steady-state behaviour that is identical to the behaviour of P. The theorem that follows captures how the next step probabilities of $\tilde{P}$ and P are equivalent.

Theorem 1 (Buchholz 1994)**.** *Consider a Markov chain with transition matrix $P \in \mathbb{R}^{N \times N}$ that is lumpable to $\Delta = \{A_1, \ldots, A_K\}$, and whose n-step state distribution is denoted by $\boldsymbol{\pi}^{(n)} \in \mathbb{R}^N$. Let $\tilde{P} \in \mathbb{R}^{K \times K}$ be a lumped matrix with entries as specified in Equation (2.26), whose n-step distribution is $\tilde{\boldsymbol{\pi}}^{(n)} \in \mathbb{R}^K$. Then if $\tilde{\pi}_l^{(0)} = \sum_{j \in A_l} \pi_j^{(0)}$, we have:*

$$\tilde{\pi}_l^{(n)} = \sum_{j \in A_l} \pi_j^{(n)}, \quad n > 0 \tag{2.27}$$

Note that we refer to the behaviour on the aggregated state-space $\{A_1, \ldots, A_K\}$. Information about the state probabilities within the classes is no longer accessible in the lumped model. That is acceptable though, as this kind of information was chosen to be ignored once the state-space was aggregated.

The concept of lumpability can be easily extended to CTMCs by considering the infinitesimal generator matrix Q instead of the probability matrix P. It is also well known that if a CTMC is lumpable to some partition Δ, the discrete-time Markov chain obtained via uniformisation is also lumpable to Δ. Lumpability is equivalent to the notion of *probabilistic bisimulation* [60] for CTMCs, as can be seen in [48].

State-space aggregation techniques that rely on this concept typically exploit the structure of some high-level description of the model. For example in [42], a lumpable partition is obtained by identifying isomorphic components of a PEPA model.

2.3.1.2 Quasi-Lumpability

In the general case, a lumpable partition might not exist. *Quasi-lumpability*, which was introduced in [34], captures approximate behaviour for Markov models. In order to describe states that exhibit approximately the same rather than identical behaviour, we have to relax the conditions in Equation (2.25).

Definition 5 (Quasi-Lumpability). *A Markov Chain with probability matrix P will be quasi-lumpable w.r.t. a partition $\Delta = \{A_1, \ldots, A_K\}$ and a bound ε, if for any two classes $A_k, A_l \in \Delta$, and for any two states $i, j \in A_k$:*

$$\left| \sum_{m \in A_l} P_{im} - \sum_{m \in A_l} P_{jm} \right| \leq \varepsilon, \quad \varepsilon \geq 0 \tag{2.28}$$

The term *near-lumpability* has been used to describe the same notion in [6], however we shall use the term "quasi-lumpability" for the rest of this thesis. Most of the research in the field so far aims at computing bounds for the state probabilities of quasi-lumpable Markov chains, assuming some partition of the state-space [34, 35, 8]. The computation of bounds of compositions of Markov chains has also been investigated in the context of Markov reward models [28] and PEPA [91].

2.3.1.3 Near Complete Decomposability

An alternative notion of approximate equivalence of states relies on the notion of *Near Complete Decomposability* (NCD) [27]. In a nearly completely decomposable Markov chain, we have highly coupled classes, while there is a relatively small probability of leaving the class. This means that a random walk will only rarely transition from one class to another. States that belong to such classes are considered to be equivalent, and this has been used in the literature as a criterion to aggregate the state-space.

In order to put this discussion in a more formal context, let us consider a Markov chain $\{X_n\}$ and a partition $\Delta = \{A_1, \ldots, A_K\}$ on its state-space. Given Markov chain state X with successor state X', we define the probability of the system moving from A_k to A_l in a single step as $Pr(X' \in A_l \mid X \in A_k)$. For a completely decomposable model we have:

$$\begin{aligned} Pr(X' \in A_k \mid X \in A_k) &= 1 \\ Pr(X' \in A_l \mid X \in A_k) &= 0, \quad \forall k \neq l \end{aligned} \tag{2.29}$$

This means that if the system is within a class A_k, it will never transition out of A_k. This condition is relaxed for nearly completely decomposable systems, where there is only a small probability of transitioning between parts of the system. More formally, the states that belong in some class A_k are approximately equivalent if:

$$\begin{aligned} &Pr(X' \in A_k \mid X \in A_k) \approx 1 \\ &Pr(X' \in A_l \mid X \in A_k) \approx 0, \quad \forall k \neq l \end{aligned} \tag{2.30}$$

2.3.1.4 A Note on the Notions of Equivalence Considered

An aggregation based on a quasi-lumpable partition $\Delta = \{A_1, \ldots, A_K\}$ implies that any information regarding the initial state probabilities is not accessible in the approximately lumped model. The only information retained is regarding the class probabilities, which are assumed to be useful enough to provide an insight into the model behaviour.

In contrast, this kind of information is not lost if the original model is nearly completely decomposable. Similarly to the quasi-lumpability case, we can calculate approximations for the class probabilities. On top of that, we can also consider the interactions within the classes in isolation [26]. That is, each class can be considered as a Markov chain that can be solved independently, and therefore we can obtain information about the original state-space probabilities.

Considering what has been stated so far, NCD is definitely more desirable as a property of the model in question. However, our objective is to employ an algorithm that blindly searches for equivalences in the state-space. The question is, which of the two properties is more likely to be found in an arbitrary Markov chain. Quasi-lumpability dictates that state-to-class probabilities for states that belong to the same class are almost the same. Under this perspective, NCD can be thought of as a special case of quasi-lumpability, since for all the states that belong to the same class, the probability of remaining in the class approaches 1, while the probabilities of transitioning to other classes approach 0. The contrary does not hold however; that is that a quasi-lumpable (or even lumpable) model does not have necessarily to be nearly-completely decomposable.

It is our opinion that quasi-lumpability covers a wider range of state equivalences, and therefore it should be more appropriate as a criterion to aggregate a Markov chain, whose properties are otherwise unknown. This opinion is evaluated experimentally later in this thesis, more specifically in Chapters 3, 4 and 6.

2.3.2 Approximate Aggregation Approaches

In this work, we discriminate the approximate aggregation approaches depending on the notion of equivalence that they rely on, which can be either quasi-lumpability or NCD. In this section we briefly outline such approaches in the literature, while a deeper presentation of the related theory is part of the discussion in Chapter 3.

Many existing approximate Markov chain aggregation techniques, such as [88, 30], rely on the notion of NCD. Methodologies that optimise the partitioning of a Markov chain with respect to this criterion typically make use of the eigen-properties of the transition probability matrix. In short, the eigenvectors that correspond to the largest eigenvalues of a stochastic matrix convey information on which of the states are strongly connected. The relation between the spectral properties of probability matrices and NCD has been investigated in a number of works [71, 81, 46]. The first fully developed approach for NCD identification can be attributed to [31], where the structure of the eigenvectors has been used to partition the state-space of reversible Markov chains, in a way that minimises the probability of transitioning between partitions. In a more recent work [30, 29], a similar approach for partitioning Markov models has been presented which is based on information theory. These techniques require that the Markov chain in question is reversible, as they exploit the symmetry imposed by the detailed balance equation. The identification of nearly-completely decomposable partitions has been extended to non-reversible models in a number of works [36, 88, 53], which make use of appropriate reversible models to approximate a non-reversible Markov chain.

It is in general more difficult to optimise the state-space partition with respect to a quasi-lumpability related metric. To the best of our knowledge, the only relevant approach in the literature has appeared in [3], where an abstract framework for this kind of optimisation is established. An algorithmic identification of lumpability is examined in [53], however aggregation of non-lumpable partitions has not been discussed.

2.4 Markov Chain Simulation

Stochastic simulation is a traditional approach for exploring the transient and steady-state properties of massive CTMCs. It simply realises a random walk over the Markov chain state-space. The output of a simulation algorithm is a *trajectory*, which involves a sequence of states, together with information on the time at which each transition happens. A collection of trajectories is then used to estimate the state probabilities at different time-points. Stochastic simulation to a random process is the equivalent of sampling to a random variable. The larger the number of the trajectories generated, the more accurate the probability estimates will be. The size of the state-space is irrelevant to the applicability of the approach, as no explicit representation of the state-space is required.

Simulation can be particularly effective for certain tasks. For example, a single (possibly long) simulation run may be sufficient to produce an estimate for the steady-state probabilities of an ergodic Markov chain, as its steady-state distribution is known to be unique. Not all the models of interest are ergodic however, neither are stationary measures always of interest. In order to obtain insights on the transient behaviour of a system with high confidence, a great number of simulation runs is typically required in the general case. There are models, especially in the context of chemical reaction networks, whose complexity renders simulation computationally expensive.

In recent years, there has been an increased interest on exploring the stochastic behaviour of biological systems via CTMC simulation. The high complexity of biological systems motivated several approaches oriented towards improving efficiency. These can be roughly divided into exact and approximate methods. The former are guaranteed to converge to the true distribution of the stochastic process, while the latter typically rely on assumptions that if they are not satisfied, a certain amount of error is introduced. In the following subsections, we review some of the approaches of each category.

2.4.1 Exact Methods

The exact simulation approaches produce trajectories that involve every single transition happening, which is the source of their high compositional cost in the case of complex models. The standard simulation method in the biological domain for CTMCs is known as the *Gillespie algorithm* or the *Direct Method* (DM) [40]. Gibson & Bruck [38, 39] proposed the *Next Reaction Method* as an efficient alternative to the DM, especially in the case of a large number of species and reaction channels.

Most of the exact simulation approaches in the literature typically involve optimisations over the DM. The *optimised direct method* was introduced in [18], where the reactions that happen most frequently are placed in the beginning of the reaction search order. A possible drawback though, is that the identification of frequent reactions requires a number of pre-simulation runs. This idea has been further developed by McCollum et al. [69], where the reaction sorting is dynamically changed throughout the simulation. Li & Petzold [64] proposed the use of an index for the reactions that permits accessing reactions in logarithmic time. Their method is called the *logarithmic direct method*.

More recent approaches stray from the path of DM optimisation. For example *ER-leap* [75] samples a number of events from a multinomial distribution whose duration is determined by a Gamma distribution. This is related to the R-leap algorithm discussed in the next section, which is essentially approximate. However, rejection sampling is used to correct the erroneous distribution. In another work [84], the notion of partial propensity function is introduced to efficiently simulate heavily coupled systems.

An approach of particular interest for this work is the *K-skip method I* [9]. This method aims at reducing the number of random samples generated, which is related to some of the contribution of this thesis. A more detailed discussion on K-skip method I can be found in Chapter 5.

2.4.2 Approximate Simulation

By definition, exact simulation algorithms require the generation of every single event happening. If those events are too many, there is a limit on how efficient an exact algorithm can be. In contrast, approximate approaches skip some simulation events, and eventually simulate a different stochastic process, which approximates the original one. This results in a significant improvement in efficiency when compared to exact methods, at the cost of an approximate solution.

One significant issue of the approximate simulation approaches is that they rely on certain assumptions that may be related to the properties of the system simulated. That means that most of the approximate methods are particularly efficient and even accurate for certain types of systems, while typically they cannot be generalised for arbitrary systems. Depending on the type of the assumptions that approximate methods depend on, we can discriminate between the two categories discussed in the rest of this section.

2.4.2.1 Time Leaping Methods

One of the first approximate simulation methods was τ-*leaping*, introduced in [41]. The τ-leaping method takes advantage of the fact that a single transition usually causes only small changes to the system. This assumption is reasonable for many biological systems, where we might not care about every single change in the molecular populations. So the system is advanced by a pre-selected time τ, during which many transitions may occur. The state of the system will be updated by taking into account all of the transitions that has happened in each leap. The value of τ should be small enough to ensure that the propensity functions do not change, and large enough at the same time, so as to result in a decent speed-up. Some of the related work involves dealing with practical considerations, such as choosing the leap size [14] and avoiding negative values for τ [15]. An alternative approach that focuses on these problems is [22], where a binomial distribution based τ-leap is employed.

The method described above however, also referred to as "explicit" τ-leaping, fails to efficiently simulate stiff systems. *Stiffness* in chemical reaction networks refers to the existence of multiple time-scales, meaning that some reactions occur significantly more frequently than others. In such cases, the τ-leap has to be set too small, in order to ensure that propensity functions do not change dramatically. In the case of a large step, the method would be unstable. The *implicit τ-leaping* method [86] deals with the stability problem of large step sizes. Despite being stable for fast components though, it fails to capture the variability of these components. Additional implicit leaps have to be inserted to restore the damped fluctuations. Another modification, *trapezoidal τ-leaping* [19], is claimed to exhibit better accuracy than the explicit and implicit methods, while it does not suffer from the damping of fast components problem. *Adaptive explicit-implicit τ-leaping* [17] switches between explicit and implicit method throughout the simulation, depending on the stiffness of the system.

In other approaches, such as *R-leaping* [1] and *K-leap* [10], stochastic simulation advances by a certain number of events. The number of events included in such a leap controls the quality of the approximation that the algorithm provides. These works are mostly different with respect to the mechanisms used for deciding the number of events to be skipped.

2.4.2.2 Time-Scale Separation Methods

The problem of stiffness, which means that the system evolves in different time-scales, motivated the development of time-scale separation methods. The approximation proposed in [85] intended to reduce the model complexity by eliminating the fast dynamics that contribute to high computational costs. This was achieved by the *quasi-steady-state assumption*, which implies that a subset of species is approximately in steady-state with respect to another time-scale.

Time-scale separation has been also exploited in the *maximal time step method* [83]. More specifically, an exact method was used for updating species with low levels of molecules, while the updates for the rest of the species were made by an approximate τ-leaping method. This procedure demanded partitioning the system into fast and slow reactions. The approach proposed in [7] expanded these ideas to multiple scales, by using a combination of the direct method, τ-leaping and solving stochastic differential equations to simulate reactions in three reaction subsets: slow, intermediate and fast.

The *slow-scale stochastic simulation* method [16, 13] is based on similar principles. Concisely, the reactions are identified as slow or fast, depending on their propensity functions. In the same way, the species that are affected by any fast reaction are also identified as fast species. Thus, the system is partitioned into two processes, fast and slow, which are definitely not Markovian, as they are dependent on each other. A Markovian *virtual fast process* is then introduced, which approximates the real fast one. On top of that, slow scale propensity functions are defined as the average of the regular ones over the fast variables. In this context, the fast variables are treated as though they were in equilibrium. This is the stochastic generalisation of the partial equilibrium approximation [87]. Eventually, the evolution of this slow-scale approximation is simulated. As noted in [12], the more the fast and the slow times are separated, the more accurate and efficient the algorithm will be. If this is not the case, the system will be not stiff and the slow-scale simulation should not be applied.

Cao & Petzold claim in [21] that slow-scale simulation has been shown to be advantageous over other methods such as adaptive τ-leaping, especially when species with small population are involved in fast reactions. Nevertheless, slow-scale's inability to cope with multiple different time scales is also identified, so they proposed a combination of the slow-scale and τ-leaping methods. According to the authors, work is still pending with respect to implementation details and theoretical issues.

Chapter 3

Approximate Markov Chain Aggregation

In a Markovian process algebra, a system is described as a collection of interacting components, whose combined state-space can be mapped to a continuous-time Markov chain. Although it is a powerful approach to construct sophisticated models using such a high-level representation, it can often result in state-space explosion. That means that the state-space of the underlying CTMC could be too large to be analysed efficiently, even if it consists of relatively simple components. In the chapter that follows the current one, we investigate the effect that component aggregation has on the global state-space. As we shall see, these components are essentially labelled CTMCs, whose state transitions are assumed not to follow any obvious pattern that can be exploited. The concept is to use a machine learning approach to detect patterns in the state-space of components. This is related to the more generic problem of Markov chain aggregation.

In this chapter, we deal with the problem of aggregating an unstructured Markov chain, intending to apply these findings to PEPA components later. Markov chain aggregation implies that the state-space is partitioned into disjoint sets of states. Ideally, this would result in a model whose transient and steady-state properties are identical to that of the original. This kind of aggregation is exact, however the resulting model will be a Markov chain only if the original model is lumpable. If the original model is not lumpable, then it is only possible to produce a Markov chain that approximates the true behaviour.

The task of approximate aggregation consists of partitioning the state-space in a way such that the approximation error is minimised, which is essentially an optimisation problem. Intuitively, we need to aggregate states that are equivalent. The notion of lumpability adequately captures state equivalence in Markov chains, as we have seen in Definition 4. Lumpability is not appropriate in an approximate aggregation context however, as a Markov chain will either be lumpable to a partition of its state-space or not. The objective is to define a computational process that converges to a partition of the state-space that is optimum with respect to some notion of approximate state equivalence. We investigate two different such notions, namely *near complete decomposability* [27] and *quasi-lumpability* [34], each of which is used to define an approximate aggregation approach.

One interesting consideration is regarding the appropriate value for the number of classes K into which a Markov chain will be partitioned. Any Markov chain with N states is trivially lumpable to a partition with $K = N$ classes that maps each state to its own class, or to a partition that maps the entire state-space to a single class. Evidently, neither of these two extremes is useful for Markov chain aggregation. The size for the aggregated model will have to be somewhere between these two extremes; it has to be significantly smaller than the original, yet informative enough to produce useful information. The selection of an appropriate size is a difficult problem that possibly requires experimentation with several values for K, guided by the experience of the modeller. Throughout this work, it is assumed that the number of classes is predetermined by the user, in a way that reflects their expectations regarding the possibility of aggregation. Given a specified number of classes, our objective is to find a partition that best approximates the original state-space.

Section 3.1 describes a partition optimisation approach that relies on the concept of near complete decomposability. An alternative partitioning approach is presented in Section 3.2, where a partition is optimised with respect to a measure that is related to quasi-lumpability. In Section 3.3, we discuss the effects of state-space aggregation, and we clarify the assumptions under which an approximately aggregated model is a Markov chain. Finally, the two aggregation approaches are experimentally evaluated in Section 3.4. We note that in most cases we make use of the embedded discrete-time Markov chain that is obtained after *uniformisation* [54].

3.1 Near-Complete Decomposability and Spectral Segmentation

We will use the term *spectral segmentation of Markov chains* to refer to a family of methods that make use of the eigenvectors of the transition probability matrix, in order to produce a partitioning of the state-space. As we shall see later in the section, such a partitioning relies on the assumption that the Markov chain is nearly completely decomposable.

Near-Complete Decomposability (NCD) is a traditional approach to define approximate state equivalence of Markov chains. To paraphrase Courtois in [26], Markov chain aggregation can be achieved if the state-space can be partitioned into classes such that:

- interactions within a class can be studied in isolation with respect to the rest of the system.
- interactions among classes can be analysed without referring to the interactions within the classes.

The first statement is true if the system is completely decomposable. This is a trivial case where parts of the system are independent. If we assume that there are only weak interactions among the classes, then the system will be nearly completely decomposable. The second statement describes a desirable aspect of such systems, which is that they can be reduced to a smaller system that involves classes and transitions among the classes only.

Definition 6 (Nearly-Completely Decomposable Stochastic Matrix). *A stochastic matrix P will be nearly completely decomposable w.r.t. a partition $\Delta = \{A_1, \dots, A_K\}$ and a bound ε, if it can be written as a sum of matrices:*

$$P = P^{-} + P^{\varepsilon}$$

where P^{-} is completely decomposable into K stochastic matrices $P_1^{-}, \dots, P_K^{-}$, or equivalently, P^{-} can be written in block diagonal form (assuming an appropriate ordering of states):

$$P^{-} = \begin{bmatrix} P_1^{-} & & & \\ & P_2^{-} & & \\ & & \cdots & \\ & & & P_K^{-} \end{bmatrix} \tag{3.1}$$

and P^{ε} contains relatively small elements whose entries have absolute values bounded by ε. Moreover, for P to be stochastic, the row sums of P^{ε} have to be equal to zero, while the entries away from the block diagonal have to be non-negative:

$$P^{\varepsilon}_{ij} \geq 0, \quad \forall i \in A_k, j \in A_l : k \neq l \tag{3.2}$$

We can now consider a Markov chain with transition probability matrix P that is nearly completely decomposable with respect to a partition $\Delta = \{A_1, \ldots, A_K\}$. According to Definition 6, it is implied that if a random walk is within a class A_k, then it tends to stay in A_k. In other words, the probability of transitioning between classes is small, while there is high probability of performing transitions within a class. More formally, given Markov chain state X with successor state X' we have:

$$\begin{aligned} Pr(X' \in A_k \mid X \in A_k) &\approx 1 \\ Pr(X' \in A_l \mid X \in A_k) &\approx 0, \quad \forall k \neq l \end{aligned} \tag{3.3}$$

where $Pr(X' \in A_l \mid X \in A_k)$ is the transition probability from A_k to A_l; we will use $Pr(A_l \mid A_k)$ as a shorthand. Note that this probability is not the same for all $i \in A_k$, as this will be:

$$Pr(X' \in A_l \mid X = i) = \sum_{j \in A_l} P_{ij} \tag{3.4}$$

Given that $\boldsymbol{\pi} > 0$ is the steady-state distribution of P, we can define the transition probability from A_k to A_l with respect to $\boldsymbol{\pi}$, which will denote the one-step probability in the long run:

$$Pr(A_l \mid A_k) = \frac{\sum_{i \in A_k, j \in A_l} \pi_i P_{ij}}{\sum_{i \in A_k} \pi_i} \tag{3.5}$$

Using the notion of NCD, we shall say that two or more states are approximately equivalent if they belong to a class A_k such that there is very small probability of transitioning out of the class in the long run. Therefore, we have to formulate an optimisation problem that selects a partition of the state-space such that the class-to-class transition probabilities as calculated by (3.5) approximate the values specified in Equation (3.3). In this section we present an approach that we call *NCD-based aggregation*.

3.1.1 Related Work on NCD Identification

The spectral properties of probability matrices is a subject well studied in the literature [27, 71, 81, 46]. The eigenstructure of a probability matrix contains information about which parts of the Markov chain are almost invariant. As can be seen in [31], a probability matrix P with K invariant subsets of states will have K eigenvalues that are equal to 1. It has been shown that states that belong to the same invariant set A_i have the same sign-structure when mapped onto the eigenvector that corresponds to eigenvalue $\lambda = 1$. Perturbation analysis that was performed in [31] shows that this property is mostly preserved for the largest K eigenvectors for a nearly completely decomposable system as well. The sign-structure of the corresponding eigenvectors has been used to identify almost invariant subsets in terms of a graph colouring algorithm.

Deng et al [30, 29] exploit almost invariant states in order to perform state-space aggregation. They proposed a recursive bi-partitioning strategy that relies on the sign-structure of the eigenvector of the probability matrix that corresponds to the second largest eigenvalue.

One important assumption of the theory developed in [31] is that the Markov chain in question has to be reversible. Non-reversible models are handled in [36], where the reversible matrix $\hat{P} = (P + \bar{P})/2$ is constructed, given a stochastic matrix P and its time-reversal $\bar{P}$. Invariant sets of states are then identified by applying a fuzzy c-means clustering algorithm [4] on the eigenvectors of $\hat{P}$. A similar approach appeared in [88], where a so-called *multiplicative reversibilisation* has been applied instead. More specifically, they apply the graph-colouring algorithm of [31] on the constructed reversible matrix $\hat{P} = P\bar{P}$. In a more recent work, Jacobi [53] relies on the spectral properties of a transformation of the original stochastic matrix in order to identify nearly-completely decomposable partitions. The transformed matrix is chosen to be self-adjoint, meaning that eigenvector calculations are robust for non-reversible Markov chains. Most of the discussion that follows assumes reversible Markov chains, while we deal with non-reversible models in a way similar to [36].

We shall see that an approach that relies on the concept of NCD is strongly related to the fundamental aspects of *spectral clustering*. Our goal is to use some results and methodologies that are well established in the field of spectral clustering, in order to apply them for Markov chain segmentation.

Regarding the methods discussed above, only Runolfsson & Ma [88], Deng et al [30, 29] and Jacobi [53] perform state-space aggregation, i.e. they replace a Markov chain with a smaller model that has approximately similar behaviour. In the current section, we are only concerned about identification of nearly-completely decomposable partitions. The discussion regarding the construction of an aggregated model is continued in Section 3.3.

3.1.2 Implications of Spectral Clustering on NCD

In short, the goal of a clustering algorithm is to identify *clusters* of "similar" instances in some input data. This data, also called the *dataset*, is often expressed as a set of points in $\mathbb{R}^N$, and similarity is subsequently defined as a function of a distance metric between instances. Although the definition of similarity is not trivial [65], it is not relevant to our discussion. Using an appropriate notion of similarity, a dataset is associated with a weighted undirected graph $G = (V,E)$ that is called the *similarity graph*. The set of vertices V corresponds to the dataset instances, while the weighted edges in E capture the pairwise similarities between those instances. Alternatively, the similarity graph is more conveniently summarised by an *affinity matrix* S, whose entries $S_{ij} > 0$ denote the pairwise similarities between the i-th and the j-th instances.

Spectral clustering involves identifying clusters in a given dataset by partitioning the underlying similarity graph. Let us consider a subset of the graph nodes $A \subseteq V$ and its complement $\bar{A}$, meaning that $A \cup \bar{A} = V$ and $A \cap \bar{A} = \emptyset$. In graph theory [32], the degree of dissimilarity of any such two sets is called the *cut*, and is equal to the total weight of the edges between the parts:

$$cut(A,\bar{A}) = \sum_{i\in A, j\in \bar{A}} S_{ij} \tag{3.6}$$

A way to partition the similarity graph is to choose a segmentation that minimises the *normalised cut* criterion [66].

$$Ncut(A,\bar{A}) = \frac{\sum_{i\in A, j\in \bar{A}} S_{ij}}{\sum_{i\in A} d_i} + \frac{\sum_{i\in \bar{A}, j\in A} S_{ij}}{\sum_{i\in \bar{A}} d_i} \tag{3.7}$$

where $d_i = \sum_{j\in V} S_{ij}$. Intuitively, Equation (3.7) implies that the partition must be such that the edges between parts of the graph are minimised and the nodes within the same clusters have a high degree of connectivity. It has been shown in [66] that the discrete solution of the graph partitioning problem above can be approximated by the solution of the following generalised eigensystem:

$$\boldsymbol{Lx} = \lambda \boldsymbol{Dx} \tag{3.8}$$

where D is a diagonal matrix with entries $D_{ii} = \sum_{j\in V} S_{ij}$, and $L = D - S$ is the graph *Laplacian*. More specifically, partitioning the graph according to the eigenvectors of (3.8) that correspond to the smallest eigenvalues approximates the minimisation of normalised cut.

3.1.2.1 The Normalised Cut Criterion for Reversible Markov Chains

Meilă & Shi [70] provide a random walks interpretation of the normalised cut criterion, which is essentially related to NCD. Given an affinity matrix S, the following stochastic matrix can be obtained:

$$P = D^{-1}S \tag{3.9}$$

where D is a diagonal matrix with entries $D_{ii} = \sum_{j \in V} S_{ij} = d_i$. Then, we have the following eigensystem:

$$P\boldsymbol{x} = \lambda \boldsymbol{x} \tag{3.10}$$

As Meilă & Shi also noticed in their work, if λ, x is an eigenvalue-eigenvector pair of the generalised eigensystem in (3.8), then $(1-\lambda)$, x are solutions of (3.10). Hence, the probability matrix P will have the same eigenvectors as the generalised graph Laplacian. This implies that the minimisation of normalised cut is approximated by the eigenvectors that correspond to the largest eigenvalues of P.

It appears that the spectral properties of probability matrices contain information that is important for clustering. This information however is also related to the notion of NCD. Using the probability matrix in (3.9), we can plug $S_{ij} = d_i P_{ij}$ into Equation (3.7) in order to obtain the following expression for the normalised cut:

$$Ncut(A,\bar{A}) = \frac{\sum_{i \in A, j \in \bar{A}} d_i P_{ij}}{\sum_{i \in A} d_i} + \frac{\sum_{i \in \bar{A}, j \in A} d_i P_{ij}}{\sum_{i \in \bar{A}} d_i} \tag{3.11}$$

Moreover, since S is symmetric, it follows that:

$$d_i P_{ij} = d_j P_{ji} \tag{3.12}$$

Now consider that P is the transition probability matrix of a Markov chain with steady-state probability vector $\boldsymbol{\pi}$. By setting the vector $\boldsymbol{d} = \boldsymbol{\pi}$, we can always construct a symmetric matrix S with entries $S_{ij} = \pi_i P_{ij}$, and Equation (3.12) tells us that P is reversible. Therefore, the entire discussion on the normalised cut is directly related to the family of reversible Markov chains. More specifically, we can rewrite (3.11) as follows:

$$Ncut(A,\bar{A}) = Pr(A \mid \bar{A}) + Pr(\bar{A} \mid A) \tag{3.13}$$

Meilă & Shi used Equation (3.13) to provide an insight on spectral clustering that relies on the random walk over the nodes of the similarity graph with affinity matrix S. That is, the minimisation of normalised cut implies minimising the probability of moving across parts of the graph in a random walk.

From a Markov chain aggregation perspective, a small value for the normalised cut between parts of the state-space implies that the probability of transitioning among parts as defined in (3.3) is small. We can see that the rationale behind spectral clustering is absolutely compatible with other Markov chain aggregation approaches that exploit the spectral properties of P. Therefore, given some reversible Markov chain with transition probability matrix P, it is possible to adapt a spectral clustering approach to obtain a partitioning of the state-space that can be considered nearly-completely decomposable.

3.1.2.2 The Non-Reversible Case

One key assumption made in the previous section is that the Markov chain is reversible. In the general case however, Equation (3.12) may not hold, which implies that the Markov chain is non-reversible and there is no symmetric matrix S associated with it. The normalised cut interpretation is no longer valid in this context.

In order to handle non-reversible models, we could construct a reversible one that shares some properties of the original non-reversible Markov chain and its time-reversal. Given some Markov process with probability matrix P and steady-state probability vector $\boldsymbol{\pi}$, its time-reversal will have transition probability matrix $\bar{P}$ with elements:

$$\bar{P}_{ij} = P_{ji}\frac{\pi_j}{\pi_i}$$

In this work, we adopt the approach of [36]; we approximate P with the following probability matrix:

$$\hat{P} = \frac{P+\bar{P}}{2} \tag{3.14}$$

In the equation above, $\hat{P}$ can be thought of as the average process of the two. It is easy to show that $\hat{P}$ is reversible with steady-state distribution $\boldsymbol{\pi}$:

$$\begin{aligned}
\pi_i\hat{P}_{ij} &= \pi_i(P_{ij} + P_{ji}\pi_j/\pi_i)/2 \\
&= (\pi_i P_{ij} + P_{ji}\pi_j)/2 \\
&= \pi_j(P_{ji} + P_{ij}\pi_i/\pi_j)/2 \\
&= \pi_j\hat{P}_{ji}
\end{aligned}$$

Assuming that we have an algorithm to identify a nearly-completely decomposable partition on $\hat{P}$, it can be easily seen that the existence of NCD in $\hat{P}$ also implies NCD in P. More formally, let $Pr(A_l \mid A_k;\ P)$ denote the long term transition probability from a class A_k to A_l for a stochastic matrix P. Then we have:

$$\begin{aligned} Pr(A_l \mid A_k;\, \hat{P}) &= \frac{\sum_{i \in A_k} \sum_{j \in A_l} \pi_i \frac{P_{ij} + \bar{P}_{ij}}{2}}{\sum_{i \in A_k} \pi_i} \\ &= 0.5\, \frac{\sum_{i \in A_k} \sum_{j \in A_l} \pi_i P_{ij}}{\sum_{i \in A_k} \pi_i} \;+\; 0.5\, \frac{\sum_{i \in A_k} \sum_{j \in A_l} \pi_i \bar{P}_{ij}}{\sum_{i \in A_k} \pi_i} \\ &= 0.5\, Pr(A_l \mid A_k;\, P) \;+\; 0.5\, Pr(A_l \mid A_k;\, \bar{P}) \end{aligned}$$

It is evident that $\forall k \neq l$, if $Pr(A_k \mid A_l;\, \hat{P}) \approx 0$ then both $Pr(A_k \mid A_l;\, P)$ and $Pr(A_l \mid A_k;\, \bar{P})$ should approach zero. Therefore, every partition for which the constructed reversible process $\hat{P}$ is nearly-completely decomposable, also implies NCD for P. Unfortunately though, this does not work both ways, as the inclusion of the reversed process $\bar{P}$ may prevent the detection of a nearly decomposable class on P.

3.1.3 NCD-based Aggregation

So far, we have seen how spectral graph segmentation is related to the notion of NCD. It is now fairly simple to adapt a spectral clustering approach, so as to define a Markov chain partitioning strategy that relies on NCD. The clustering algorithm of our choice is the one proposed by Ng et al in [78], for reasons that will be made clear later in this section. Recall that in spectral clustering the data is partitioned according to the eigenvectors of the Laplacian matrix. In [78], the following symmetric version of the Laplacian matrix is used:

$$L_{sym} = D^{-1/2} S D^{-1/2} \tag{3.15}$$

By simple linear algebra manipulations, it can be easily shown that P and L_{sym} are related as follows:

$$L_{sym} = D^{1/2} P D^{-1/2} \tag{3.16}$$

In fact L_{sym} and P are similar, which means that they have the same eigenvalues. It can also be seen that their eigenvectors are also related; if $\boldsymbol{x}$ is an eigenvector of P, then $D^{1/2}\boldsymbol{x}$ will be an eigenvector of L_{sym}. The Markov chain adaptation of the Ng et al method is summarised in the steps of Algorithm 1.

The final step of Algorithm 1 clusters the data according to their affinities when mapped onto the space spanned by the K largest eigenvectors of the Laplacian L_{sym}. This process of clustering is described in a rather abstract way as the output of a K-*means* clustering approach. This step requires further explanation, as K-means cannot

Algorithm 1 NCD-based Markov Chain Aggregation

```
1: Consider a Markov chain with transition matrix $P \in \mathbb{R}^{N\times N}$
2: Given $\bar{P}$, which is the time-reversal of $P$, construct the reversible Markov chain
   with transition probability matrix:
       $$\hat{P} = \frac{P+\bar{P}}{2}$$
3: Construct the diagonal matrix $D$ with entries $D_{ii} = \pi_i$, where $\boldsymbol{\pi}$ is the steady-state
   probability vector of $\hat{P}$
4: Construct the symmetric Laplacian:
       $$L_{sym} = D^{1/2}\hat{P}D^{-1/2}$$
5: Find the eigenvectors that correspond to the $K$ largest eigenvalues of $L_{sym}$ and form
   the matrix:
       $$X = [\boldsymbol{x_1}\boldsymbol{x_2}\dots\boldsymbol{x_K}] \in \mathbb{R}^{N\times K}$$
6: Normalise each row of $X$ so as it has unit length:
       $$Y_{ij} = \frac{X_{ij}}{\sqrt{\sum_{j=1}^{K} X_{ij}^2}}$$
7: Given that each row of $Y$ is a data-point in $\mathbb{R}^K$, perform a K-means clustering
```

guarantee a globally optimal partition. In most implementations, it starts from a random initial solution and performs a number of iterations until it converges to a local optimum [93]. In practice, multiple runs are required to obtain a globally optimal solution.

The cost of performing K-means has been significantly reduced in terms of the Ng et al method [78] using an appropriate initialisation. Note that a normalised version of the eigenvectors is used, which are stored as columns in the Y matrix. In the ideal case, that is when the Markov chain is completely decomposable, the points $Y_i \in \mathbb{R}^K$ have the form $(0,\dots,0,1,0,\dots,0)$ where the position of the "1" indicates the connected component this point belongs to. Notice that instances that belong to different clusters are orthogonal to each other. Perturbation analysis that was performed in [78] shows that this property is mostly preserved for the largest K eigenvectors for a nearly completely decomposable system as well. The idea is to select an initial solution in the form of K centroids, where each centroid is a row of Y. Starting from any row of Y, we repeatedly choose the next row that is closer to being perpendicular to all the centroids chosen so far. The angle between different rows of Y can be trivially measured by means of the inner product; obviously we need to select such a row of Y that the inner product with the rest of the centroids is as close to zero as possible.

It is claimed in [78] that the subspace spanned by the first K eigenvectors of L_{sym} will be stable to small changes of L_{sym}, if and only if the *eigengap* $\delta = |\lambda_K - \lambda_{K+1}|$ is large. Ideally, for a matrix that is nearly-completely decomposable to K classes, the first K eigenvalues should be close to 1, while the rest of the eigenvalues should be bounded away from 1. If the data is not well separated in this way, that simply means that the Markov chain is not really nearly decomposable into any partition of K classes. The resulting partition in that case would be one that just approaches NCD, without really meaning that NCD is achieved, as this would only be true if the K largest eigenvalues are close to 1.

One limitation of NCD-based aggregation is that it requires knowledge of the steady-state distribution vector $\boldsymbol{\pi}$, at the third step of Algorithm 1. In practice, its calculation negates any requirement to aggregate the model, unless we are interested in the transient behaviour only. Nevertheless, we shall see in Chapter 4 that this is a small price to pay if the aggregation is applied in a compositional setting.

3.2 The Quasi-lumpability Approach

One of the main contributions of this thesis is to propose a partitioning strategy for Markov chains which complies with a notion of approximate state equivalence. We have seen that most existing approximate aggregation approaches implicitly rely on the notion of NCD, however we think that quasi-lumpability covers a wider range of approximate equivalences. In this section, we investigate the possibility of defining Markov chain partitioning as an optimisation task that relies on the concept of quasi-lumpability. We introduce a measure that is related to quasi-lumpability and we discuss how this measure can be used in terms of a partitioning strategy. The method that we propose involves formulating Markov chain partitioning as a clustering problem.

3.2.1 A Pseudo-metric related to Quasi-Lumpability

As we have seen in Definition 5, considering a Markov chain with probability matrix P that is quasi-lumpable w.r.t. $\Delta = \{A_1, \ldots, A_K\}$, then for any two classes $A_k, A_l \in \Delta$ and for any two states $i, j \in A_k$ we have:

$$\left| \sum_{m \in A_l} P_{im} - \sum_{m \in A_l} P_{jm} \right| \leq \varepsilon, \quad \varepsilon \geq 0$$

The equation above has simply resulted from relaxing the conditions imposed for lumpable Markov chains in Definition 4. The quantity ε in the equation above corresponds to the maximum difference between elements that are assigned to the same class. If we consider the transition probability matrix P of a quasi-lumpable model, this can be represented as $P = P^- + P^\varepsilon$, where P^- is a lumpable Markov chain and P^ε is a matrix whose entries have absolute values bounded by the ε quantity of Equation (2.28). In general, most of the values of P^ε should be zero, while the non-zero elements should be small. As noted in [6], if ε is sufficiently small, the lumpable model with transition matrix P^- approximates the behaviour of the quasi-lumpable one.

Using Equation (2.28), we can define a pseudo-metric that captures a similarity distance between states. If we consider all classes $A_1, \ldots, A_K$, we define the following quantity for any two states i, j that belong to the same class:

$$E_{i,j} = \sum_{l=1}^{K} \left| \sum_{m \in A_l} P_{im} - P_{jm} \right| \tag{3.17}$$

In the equation above, $E_{i,j}$ will be equal to zero, iff the Markov chain is lumpable with respect to the partition $\Delta = \{A_1, \ldots, A_K\}$. Since it is possible that $E_{i,j} = 0$ when $i \neq j$, $E_{i,j}$ is characterised as a pseudo-metric, rather than as a metric.

Hence, the optimal quasi-lumpable partition will be the one that minimises the quantity $E_{i,j}$ for any two states in the same class. However, the value of $E_{i,j}$ depends not only on the transition probabilities of states i and j, but also on the way that the states are distributed across the classes. In other words, a different partitioning of the state-space will result in a completely different $E_{i,j}$ quantity for the very same i and j states. Thus, it is very difficult to design an algorithm that minimises $E_{i,j}$ with respect to the partitioning.

3.2.2 Formulation as a Clustering Problem

According to Definition 5, we can think of approximate state-space aggregation as a problem of minimising the ε quantities with respect to the partition selected. Since this is a problem of clustering states with respect to a measure, it is rather intuitive to turn towards clustering algorithms [93]. By definition, algorithms of this kind produce clusters of the input data, such that the distance between objects that belong in the same cluster is minimised.

The pseudo-metric $E_{i,j}$ that measures dissimilarity between states is not appropriate to be directly used by a clustering algorithm. The $E_{i,j}$ values are not constant for each pair of states, as they depend on the partition. Instead, we show that the pseudo-metric $E_{i,j}$ is bounded by a proper distance metric that is independent of the partitioning, and clustering can be therefore applied in this context.

Starting from Equation (3.17), if we pull the inner sum out of the absolute value, we will have a larger value according to the triangle inequality:

$$E_{i,j} \leq \sum_{l=1}^{K} \sum_{m \in A_l} \left| P_{im} - P_{jm} \right| \tag{3.18}$$

It is evident that the sums in the inequality above cover the entire state-space of the original Markov model. Thus, given that the initial model has N states, the right-hand side of the inequality above can be written as:

$$D_{i,j} = \sum_{n=1}^{N} \left| P_{in} - P_{jn} \right| \tag{3.19}$$

which is actually the *Manhattan distance* in the $\mathbb{R}^N$ space defined by the transition probabilities. To put it differently, we consider the states as N-valued vectors, where each one of the values is a transition probability to another state.

This shows that $D_{i,j} \geq E_{i,j}$. It is relatively straightforward to apply a clustering algorithm in order to identify K clusters such that the Manhattan distance $D_{i,j}$ is minimised for instances that belong to the same cluster. The minimisation of $D_{i,j}$ will result in small values for $E_{i,j}$, and hence for the ε quantity in the quasi-lumpability definition as well.

3.2.2.1 The Clustering Algorithm

In order to obtain a partitioning of the state-space that minimises the Manhattan distance for states in the same cluster, we have to apply a clustering algorithm. Typical clustering techniques, such as K-means or *Expectation-Maximisation* [5, 93], start from a randomly-picked initial solution and they perform a number of iterations until they converge to some optimum. Typically, multiple runs are required, as the solution obtained at each run is dependent on the initial randomly-picked solution.

In contrast, spectral clustering [66, 78, 65] implies that a dataset is partitioned depending on the eigenvectors of the Laplacian matrix, rather than on the local proximities of data-points. Concisely, the K eigenvectors that correspond to the largest K eigenvalues of the Laplacian are selected. The data is mapped to the rows of the $N \times K$ matrix formed by stacking these eigenvectors as columns. The clusters of data are well separated in this $\mathbb{R}^K$ space, meaning that it should be easy to identify a globally optimal clustering, in contrast to "conventional" clustering techniques whose solutions are only locally optimal. As different authors use different versions of the Laplacian matrix, there are several different interpretations of why this approach is successful [66, 78, 77].

Our approach for Markov Chain Aggregation based on quasi-lumpability makes use of the algorithm proposed by Ng et al in [78]. Although that is the same algorithm that we have used for NCD-based aggregation in Section 3.1.3, the effect is different, as the states are partitioned with respect to their pairwise Manhattan distances, rather than the transition matrix P. Other than that, the algorithm is identical to Algorithm 1, including the K-means step, which clusters the data according to their affinities when mapped onto the space spanned by the normalised eigenvectors of L_{sym}.

Step 2 of Algorithm 2 involves the computation of a similarity measure between two states given their distance. As we can see, the definition of similarity depends on the parameter σ^2, which controls how quickly the affinity S_{ij} degrades. In fact, σ^2 affects how wide a cluster can be, i.e. large values for σ^2 favour wide clusters. As noted in [65], spectral clustering algorithms do not make strong assumptions on the form of the clusters. A too large value for σ^2 could lead to cluster shapes that are not meaningful in terms of quasi-lumpability, where we would only want to minimise the Manhattan distance between each pair of states in the same cluster. Therefore, we need a value for σ^2 that imposes "tight" clusters. Since we use stochastic matrices only (the generator matrices of CTMCs are uniformised), we know that the maximum Manhattan distance between any two rows is 2. In Figure 3.1 we can see the effect of different σ^2 values on the similarity function. An appropriate value to enforce tight clusters is 0.1, which has been used in the experiments later in this thesis.

3.2.3 Spectral Properties and Quasi-Lumpability

We conclude this section by making some comments on how spectral properties of Markov chains contain information about lumpable partitions. A vector $\boldsymbol{x}$ is defined to

Algorithm 2 The Quasi-Lumpability Approach

```
1: Consider a Markov chain with transition matrix $P \in \mathbb{R}^{N\times N}$
2: Compute the affinity matrix $S \in \mathbb{R}^{N\times N}$ with entries:
       $$S_{ij} = \exp\left(-\frac{\sum_{n=1}^{N} |P_{in} - P_{jn}|}{2\sigma^2}\right)$$
3: Construct the diagonal matrix $D$ with entries $D_{ii} = \sum_{j=1}^{N} S_{ij}$
4: Construct the symmetric Laplacian:
       $$L_{sym} = D^{-1/2} S D^{-1/2}$$
5: Find the eigenvectors that correspond to the $K$ largest eigenvalues of $L_{sym}$ and form
   the matrix:
       $$X = [\boldsymbol{x_1 x_2} \ldots \boldsymbol{x_K}] \in \mathbb{R}^{N\times K}$$
6: Normalise each row of $X$ so as it has unit length:
       $$Y_{ij} = \frac{X_{ij}}{\sqrt{\sum_{j=1}^{K} X_{ij}^2}}$$
7: Given that each row of $Y$ is a data-point in $\mathbb{R}^K$, perform a K-means clustering
```

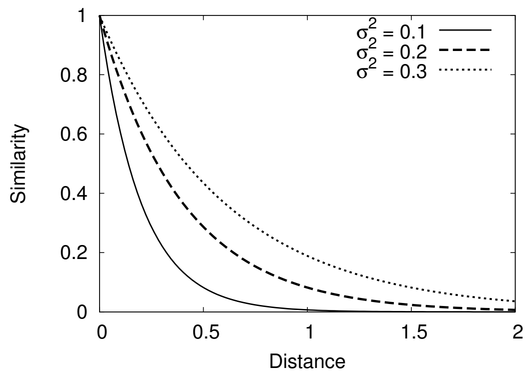


Figure 3.1: The similarity function $\exp\left(-\frac{x}{2\sigma^2}\right)$ for different values of σ^2

be *piecewise constant* w.r.t. a partition $\Delta = \{A_1, \ldots, A_K\}$, if $x_i = x_j$ for i, j indices in the same class $A_k \in \Delta$. The following proposition, found in [70], relates some properties of the eigenvectors of P with lumpability.

Proposition 1 (Meilă & Shi 2001). *Let P be the transition probability matrix of a Markov chain. Let $\Delta = \{A_1, \dots, A_K\}$ be a partition of the state-space. Then, P has K eigenvectors that are piecewise constant w.r.t. Δ and correspond to non-zero eigenvalues of P, iff the sums $\tilde{P}_{il} = \sum_{j \in A_l} P_{ij}$ are constant $\forall\ i \in A_K$ and all $k, l = 1, \dots K$ and the matrix $\tilde{P}$ as defined in Equation (2.26) is non-singular.*

The proposition above implies that given a probability matrix P, there are K independent eigenvectors of P that are piecewise constant w.r.t. a partition Δ, if and only if the corresponding Markov chain is lumpable with respect to the partition Δ and the lumped $\tilde{P}$ is not singular. Therefore, for the lumpable case we know that at most K eigenvectors are piecewise constant with respect to the lumpable partition, hence they contain information about how the states of a Markov chain can be aggregated to form a lumped model.

If we now relax this definition to allow almost piecewise constant eigenvectors, we can obtain a quasi-lumpable model. However, we have no information about which K of the eigenvectors have the property of being piecewise constant. If we just select the eigenvectors of P that correspond to the largest eigenvalues, we only minimise the normalised cut. As we have seen in the previous subsection, this implies NCD rather than quasi-lumpability.

The biggest issue of any effort to infer a lumpable partition from the eigenvectors is that there is no way to decide which of the eigenvectors contain the information that is most useful for identifying a lumpable or a quasi-lumpable partition. We think however that any attempt to distinguish systematically the appropriate eigenvectors could be an interesting direction for future research.

3.3 Constructing the Aggregated Model

We have discussed so far how to obtain a partitioning of the state-space that is nearly optimal with respect to a notion of equivalence of states. The next step is to redeem such a partition in order to construct a Markov chain that approximates the original model. In this section, we devise an approximate lumping strategy, and we show that the aggregated models lie within the bounds calculated by stochastic comparison of Markov chains.

3.3.1 Approximate Lumping

We shall now assume a Markov chain with transition matrix P that is not lumpable to $\Delta = \{A_1, \ldots, A_K\}$. Given Markov chain state X with successor state X', the probability of transitioning from state i to a class A_l will be:

$$Pr(X' \in A_l \mid X = i) = \sum_{j \in A_l} P_{ij} \tag{3.20}$$

Since P is not lumpable, then $Pr(X' \in A_l \mid X = i)$ will not be the same for different $i \in A_k$. Thus we cannot simply summarise the transition probability from A_k to A_l with a single number as done in Equation (2.26). The A_k class is supposed to form a single state in the lumped process, which is essentially non-Markovian because the transition probabilities depend on factors other than the current state alone.

3.3.1.1 Weighted Lumping

In order to produce a lumped Markov chain that approximates the behaviour of the original model, we need a way to summarise these transitions in a meaningful way. The only requirement is that each class-to-class transition should be represented by a single probability. Considering a probability vector $\boldsymbol{w} \in \mathbb{R}^N$, we can define an approximately lumped matrix $\tilde{P}$ with entries:

$$\tilde{P}_{kl} = \frac{\sum_{i \in A_k} \sum_{j \in A_l} w_i P_{ij}}{\sum_{i \in A_k} w_i} \tag{3.21}$$

Using the vector $\boldsymbol{w}$ in (3.21), we calculate the weighted average for different state-to-class probabilities, for states that belong in the same class. We shall refer to this kind of aggregation as *weighted lumping*.

In the lumpable case, the result of (3.21) does not really depend on the weighting vector. For a non-lumpable model though, we have to choose such a $\boldsymbol{w}$ vector that the aggregated matrix $\tilde{P}$ best approximates the original Markov chain P. The theorem that follows will give rise to a very interesting argument with respect to what can be considered as an optimal choice for $\boldsymbol{w}$.

Theorem 2. *Consider a Markov chain with transition matrix $P \in \mathbb{R}^{N \times N}$, whose n-step state distribution is denoted by $\boldsymbol{\pi}^{(n)} \in \mathbb{R}^N$. Given a partition $\Delta = \{A_1, \ldots, A_K\}$ on P, let $\tilde{P} \in \mathbb{R}^{K \times K}$ be a lumped matrix with entries as specified in Equation (3.21), whose n-step distribution is $\tilde{\boldsymbol{\pi}}^{(n)} \in \mathbb{R}^K$. If the current step probabilities $\tilde{\boldsymbol{\pi}}^{(n)}$ are exact, then $\tilde{\boldsymbol{\pi}}^{(n+1)}$ will also be exact if $\boldsymbol{w} = \boldsymbol{\pi}^{(n)}$.*

Proof. We assume that the current step probabilities are exact:

$$\tilde{\pi}_l^{(n)} = \sum_{j\in A_l} \pi_j^{(n)}$$

We want to see under which conditions the next step probabilities will be also exact, if we produce an aggregated matrix $\tilde{P}$ using Equation (3.21). The next step probabilities for $\tilde{P}$ will be:

$$\begin{aligned}
\tilde{\pi}_l^{(n+1)} &= \sum_{k=1}^{K} \tilde{\pi}_k^{(n)} \tilde{P}_{kl} \\
&= \sum_{k=1}^{K} \tilde{\pi}_k^{(n)} \frac{\sum_{i\in A_k}\sum_{j\in A_l} w_i P_{ij}}{\sum_{i\in A_k} w_i} \\
&= \sum_{k=1}^{K} \sum_{i\in A_k} \left(\pi_i^{(n)}\right) \frac{\sum_{i\in A_k}\sum_{j\in A_l} w_i P_{ij}}{\sum_{i\in A_k} w_i}
\end{aligned}$$

If we want $\tilde{\boldsymbol{\pi}}^{(n+1)}$ to be exact, it is necessary that we select $\boldsymbol{w}$ such that it is equal to the true aggregated probabilities $\sum_{i\in A_k} \pi_i^{(n)} = \sum_{i\in A_k} w_i$. Then we will have:

$$\tilde{\pi}_l^{(n+1)} = \sum_{k=1}^{K} \sum_{i\in A_k} \sum_{j\in A_l} w_i P_{ij}$$

The last equation implies that $\tilde{\boldsymbol{\pi}}^{(n+1)}$ will be exact if $\boldsymbol{w} = \boldsymbol{\pi}^{(n)}$, as we will have:

$$\begin{aligned}
\tilde{\pi}_l^{(n+1)} &= \sum_{k=1}^{K} \sum_{i\in A_k} \sum_{j\in A_l} \pi_i^{(n)} P_{ij} \\
&= \sum_{i=1}^{N} \sum_{j\in A_l} \pi_i^{(n)} P_{ij} \\
&= \sum_{j\in A_l} \sum_{i=1}^{N} \pi_i^{(n)} P_{ij} \\
&= \sum_{j\in A_l} \pi_j^{(n+1)}
\end{aligned}$$

□

In fact, the next step probabilities can be accurately calculated for $\tilde{P}$ if the current state distribution of P is known. There are two problems with this statement: the first is that we have to know $\boldsymbol{\pi}$, rather than the aggregated probabilities $\tilde{\boldsymbol{\pi}}$, and the second problem is that $\tilde{P}$ has to be recalculated for every different weighting vector to get the correct result. The main point of Theorem 2 is that there is no weighting vector $\boldsymbol{w}$ which can capture the exact distribution of P at any step. It is possible that there might exist some other $\boldsymbol{w}$ that results in a better approximation for the majority of the distributions, which however cannot be obtained unless P is extensively solved for its transient and steady-state probabilities.

3.3.1.2 Ideal Aggregate

We shall comment on a special case of weighted lumping, where the weighting vector is chosen to be the steady-state distribution $\boldsymbol{\pi}$ of the original transition matrix P. In the literature we can find examples where this $\boldsymbol{\pi}$-weighted lumping has been used [3, 30, 88, 53]. This kind of lumped matrix was identified as *ideal aggregate* in [6].

$$\tilde{P}_{kl} = \frac{\sum_{i\in A_k}\sum_{j\in A_l}\pi_i P_{ij}}{\sum_{i\in A_k}\pi_i} \tag{3.22}$$

Ideal aggregation captures the fact that in the long run certain states are more probable than others in the same class, and the class-to-class transition probabilities are modified accordingly. This kind of adjustment results in the approximately lumped matrix $\tilde{P}$ produced by Equation (3.22), whose steady-state probabilities $\tilde{\boldsymbol{\pi}}$ are exact, that is:

$$\tilde{\pi}_l = \sum_{j\in A_l}\pi_j \tag{3.23}$$

The proof of this statement simply follows from Theorem 2, if we consider $\boldsymbol{\pi}^{(n)} = \boldsymbol{\pi}$. Nevertheless, this property is not particularly useful, since the steady-state vector $\boldsymbol{\pi}$ is pre-calculated for the P matrix. Moreover, we think that this kind of aggregation is not appropriate for approximating the transient behaviour, as a strong bias towards the steady state behaviour is inherently introduced.

3.3.1.3 Uniform Lumping

Considering this discussion on weighting vectors, we think that it is most appropriate that $\boldsymbol{w}$ is uniformly distributed, meaning there is no bias among the states in the same class. As a matter of fact, a uniform $\boldsymbol{w}$ captures our ignorance about the state distribution. The entries of the approximately lumped matrix $\tilde{P}$ can then be more simply calculated as follows:

$$\tilde{P}_{kl} = \frac{\sum_{i\in A_k}\sum_{j\in A_l}P_{ij}}{|A_k|} \tag{3.24}$$

where $|A_k|$ denotes the number of states included in class A_k. In the case of quasi-lumpable models in particular, the $Pr(A_l \mid i)$ probabilities will be approximately the same for $i \in A_k$. The mean value is a reasonable approximator for populations characterised by almost the same value. The quality of the approximation is solely dependent on the partition. It is evident that in the lumpable case, Equation (3.24) degrades to Equation (2.26), as it would for any weighting vector.

As we show next, one important aspect of this *uniform lumping* strategy is that it produces aggregated Markov chains whose behaviour lie within the stochastic bounds that can be obtained for the original model. This is more extensively discussed in the section that follows, where we also introduce some concepts regarding stochastic comparison of Markov chains.

3.3.2 Stochastic Bounds

We know that the behaviour of an approximately lumped matrix $\tilde{P}$ produced by Equation (3.24) is definitely not equivalent to the original Markov chain P, unless P is lumpable to the partition Δ. In a different case, the calculated transient and steady-state probabilities will only approximate the probabilities in the true model. Such approximate methods are typically more useful if they can guarantee that the estimated measure is within certain bounds.

Stochastic comparison of Markov chains provides bounds for both transient and stationary measures. Knowing that P is not lumpable to Δ, the idea is to produce stochastic matrices L and U that will serve as lower and upper bound for P correspondingly. If both L and U are lumpable to Δ, then we can accurately use their lumped versions to obtain the desired bounds.

3.3.2.1 Stochastic Comparison of Random variables

To further explain stochastic comparison, we have to introduce some definitions.

Definition 7 (Stochastic Order)**.** *If X and Y are random variables that take values on an ordered state-space S, then X is said to be less than Y in the strong stochastic sense, that is $X <_{st} Y$, iff for all $k \in S$:*

$$Pr(X > k) \;\leq\; Pr(Y > k) \tag{3.25}$$

A stochastic order describes the concept of a random variable Y taking values that are most probably higher than the values of a random variable X. Following Definition 7, we can obtain an upper bound for the cumulative probability distribution of X. Thus, if $X <_{st} Y$ we have:

$$Pr(X \leq k) \;\geq\; Pr(Y \leq k) \tag{3.26}$$

Assuming that we need to approximate a random variable X, we have to define random variables Y, Z such that $Z <_{st} X <_{st} Y$, so as to produce bounds for the cumulative probabilities of X:

$$Pr(Y \leq k) \ \leq \ Pr(X \leq k) \ \leq \ Pr(Z \leq k) \tag{3.27}$$

Finally, using Equation (3.27) we can obtain bounds for the probability values. A lower bound for the value of $Pr(X = k)$ will be:

$$\begin{aligned} Pr(X = k) &\geq Pr(Y \leq k) - Pr(X < k) && (3.28) \\ &\geq Pr(Y \leq k) - Pr(Z < k) && (3.29) \end{aligned}$$

since $Pr(X < k)$ cannot be larger than $Pr(Z < k)$ according to (3.27). In the same way, an upper bound for the probability $Pr(X = k)$ will be:

$$\begin{aligned} Pr(X = k) &\leq Pr(Z \leq k) - Pr(X < k) && (3.30) \\ &\leq Pr(Z \leq k) - Pr(Y < k) && (3.31) \end{aligned}$$

since $Pr(X < k)$ is bounded by $Pr(Y < k)$.

3.3.2.2 Stochastic Comparison of Markov Chains

In practice we have a Markov process $\{X_t\}$, and we want to obtain two lumpable Markov processes $\{Y_t\}$ and $\{Z_t\}$ such that $\{Z_t\} <_{st} \{X_t\} <_{st} \{Y_t\}$. That is the state distribution of X_t should be bounded by Y_t and Z_t for any $t \geq 0$. For simplicity, we shall only discuss the case of DTMCs, as the same results apply for CTMCs after uniformisation is performed. In the current section, we briefly describe the approach used in [33] to produce lumpable bounds on Markov chains, which we shall use in the experiments of Section 3.4. The method relies on Theorem 3, which itself makes use of the notions of *comparability* and *monotonicity* for stochastic matrices defined below. Note that in the definition of comparability, the "$<_{st}$" operator has been overloaded to also cover stochastic comparison between stochastic matrices.

Definition 8 (Comparability)**.** *If P and P' are stochastic matrices, then we shall say that $P <_{st} P'$ iff for all i, we have $P_{i*} <_{st} P'_{i*}$ (we consider the rows of P and P' as vectors).*

Definition 9 (Monotonicity)**.** *Let P be a stochastic matrix, then P is st-monotone iff for any of the probability vectors x, y, if $x <_{st} y$ then $xP <_{st} yP$.*

Comparability enables us to perform stochastic comparison of Markov chains by simply comparing their transition probability matrices. Knowing however that P is less than P' in the strong stochastic sense is not adequate to assume the same for their corresponding Markov chains. It is also requires that either P or P' is monotone, according to the theorem below, whose proof can be found in [33].

Theorem 3. *Let $\{X_n\}$ and $\{Y_n\}$ be DTMCs with transition probability matrices P and P' correspondingly. Then $\{X_n\} <_{st} \{Y_n\}$ if:*

- *$X_0 <_{st} Y_0$,*
- *$P <_{st} P'$,*
- *either P or P' is st-monotone.*

Thus following Theorem 3, in order to bound the behaviour of a non-lumpable matrix P, we have to construct stochastic matrices L and U such that $L <_{st} P <_{st} U$ where both L and U are chosen to be monotone. If both L and U are lumpable to Δ, then we can accurately use their lumped versions $\tilde{L}$ and $\tilde{U}$, which will serve as lower and upper bound for P correspondingly.

3.3.2.3 Bounds for Approximately Lumped Markov Chains

We shall see that the approximately lumped matrix $\tilde{P}$ whose entries are given by Equation (3.24) is related to the stochastic bounds. In fact, if P is bounded by L and U, then $\tilde{P}$ is also bounded by the lumped versions of those matrices, as captured in the following proposition.

Proposition 2. *Consider stochastic matrices P, L and U such that $L <_{st} P <_{st} U$. Let $\Delta = \{A_1, \ldots, A_K\}$ be a partition that is lumpable for L and U but not for P. If $\tilde{L}$ and $\tilde{U}$ are the lumped versions of L and U correspondingly, and $\tilde{P}$ is the approximately lumped version of P w.r.t. Δ, then $\tilde{L} <_{st} \tilde{P} <_{st} \tilde{U}$.*

Proof. We shall prove the proposition for the upper and the lower bound independently. Since $P <_{st} U$, for each row we have $P_{i*} <_{st} U_{i*}$, which implies that $\forall \alpha \in \{1, \ldots, N\}$ we have:

$$\sum_{j=\alpha}^{N} P_{ij} \leq \sum_{j=\alpha}^{N} U_{ij}$$
$$\sum_{j=1}^{\alpha} P_{ij} \geq \sum_{j=1}^{\alpha} U_{ij}$$

Let β be a class index such that $a \in A_\beta$. We can then break the summation over j into two summations: over classes of Δ and over instances within the classes:

$$\sum_{l=1}^{\beta} \sum_{j \in A_l} P_{ij} \geq \sum_{l=1}^{\beta} \sum_{j \in A_l} U_{ij}$$

The inequality above is true for any row. We can rewrite the inequality by considering the lumped version of U:

$$\sum_{l=1}^{\beta} \sum_{j \in A_l} P_{ij} \geq \sum_{l=1}^{\beta} \tilde{U}_{kl}, \quad \forall i \in A_k$$

We can sum both parts over $i \in A_k$. Note that $\sum_{l=1}^{\beta} \tilde{U}_{kl}$ is independent of $i \in A_k$, so we simply multiply by the class size $|A_k|$:

$$\sum_{l=1}^{\beta} \sum_{i \in A_k} \sum_{j \in A_l} P_{ij} \geq |A_k| \sum_{l=1}^{\beta} \tilde{U}_{kl}$$

We can divide both parts by the size of the A_k class. Then P can be replaced by the approximately lumped version given by Equation (3.24). Hence, we have:

$$\sum_{l=1}^{\beta} \tilde{P}_{kl} \geq \sum_{l=1}^{\beta} \tilde{U}_{kl}$$

which implies that $\tilde{P}_{k*} <_{st} \tilde{U}_{k*}$, and therefore $\tilde{P} <_{st} \tilde{U}$.

Regarding the lower bound, starting from $L_{i*} <_{st} P_{i*}$ we can use exactly the same steps to show that $\tilde{L} <_{st} \tilde{P}$. □

In the experiment of Section 3.4.2 we demonstrate an example of how the behaviour of a uniformly lumped model is contained within the stochastic bounds. The bounding algorithm used is LIMSUB [33], whose acronym stands for *lumpable irreducible monotone stochastic upper bounding*. We note however that the implication of Proposition 2 is that any uniformly lumped Markov chain will be bounded, regardless of the bounding algorithm.

Nevertheless, that does not automatically imply that we have a good approximation. Even a poor approximation will still be within the bounds obtained by stochastic comparison of Markov chains, but that is because these bounds are simply too wide to provide useful information. In fact, approximation quality depends on the partition. A good partition provides an accurate approximation which is ideally accompanied by tight bounds.

3.4 Experiments and Discussion

In this final section, we perform a series of experiments in order to see how approximate aggregation behaves. The main task is to evaluate and compare the approximation quality of the two approaches that we discuss in this chapter: NCD-based aggregation (Algorithm 1) and quasi-lumpability-based aggregation (Algorithm 2). Moreover, we also investigate the effects that these approaches have on the calculation of lumpable stochastic bounds.

3.4.1 Comparison of Aggregation Approaches

In order to evaluate approximation quality, we have to compare the behaviour of an approximately aggregated Markov chain with the behaviour of the true model. This can be achieved by comparing the steady-state and the transient distributions between the original and the reduced model.

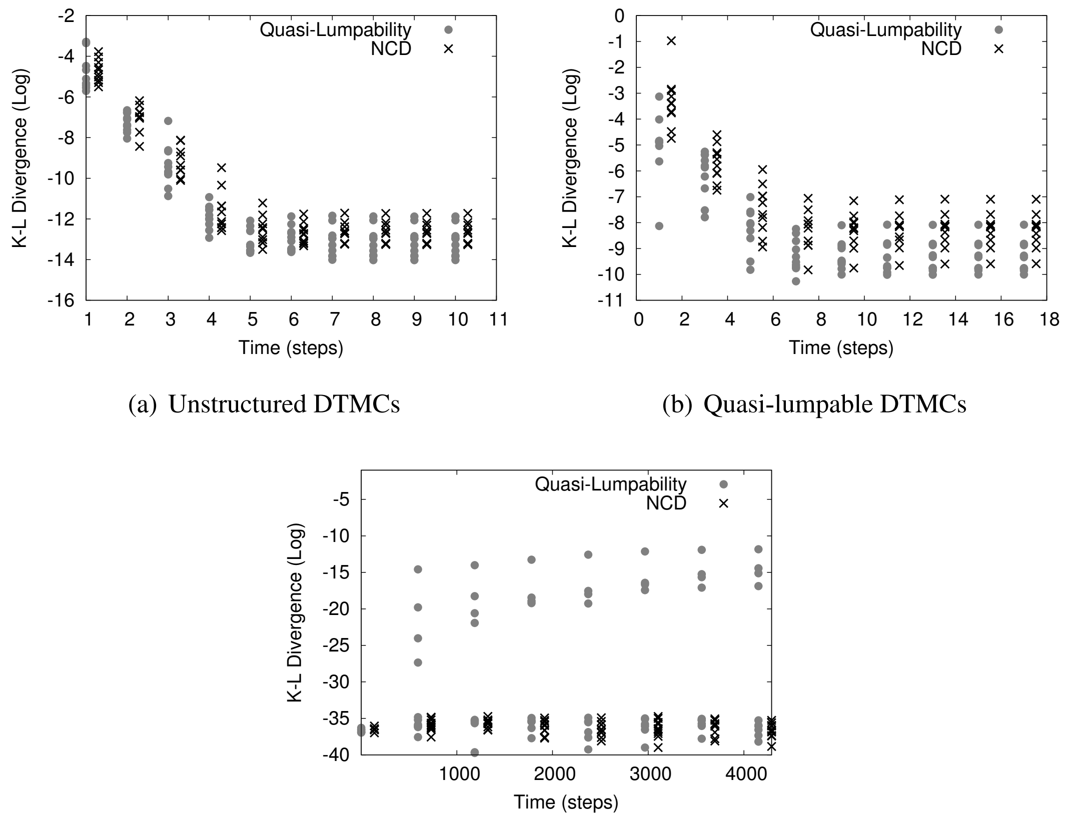
The *K-L divergence* is a very popular measure for comparing probability distributions. For two probability vectors $\boldsymbol{p}$ and $\boldsymbol{q}$, it is defined as:

$$KL(\boldsymbol{p}||\boldsymbol{q}) = \sum_i p_i \log \frac{p_i}{q_i} \tag{3.32}$$

Given a partition of the state-space with K classes, we define $\boldsymbol{p}$ as a K-valued vector containing the aggregated probabilities of the original system according to the partition of the state-space used. Then, $\boldsymbol{q}$ will be a K-valued vector containing the probabilities of the corresponding reduced model, which is produced by either the quasi-lumpability or the NCD-based approach.

It is known that $KL(\boldsymbol{p}||\boldsymbol{q}) \geq 0$, where the equality holds if, and only if, $\boldsymbol{p} = \boldsymbol{q}$. For a good approximation, the K-L divergence from the original state distribution should approach zero. However, that should not be used as an absolute measure of quality, as it depends on the distributions that are compared. Given that any comparison is made with respect to the same original state distribution $\boldsymbol{p}$, we can produce comparative results; that is to see which one of the approximation methods results in the lowest K-L divergence from $\boldsymbol{p}$.

Figure 3.2 summarises the results for a number of experiments. Each experiment involves a randomly generated DTMC featuring 400 states, whose state-space is partitioned using either the NCD-based or the quasi-lumpability approach. The model is then approximately lumped to 100 states according to Equation (3.24). The values plotted are the K-L divergences at different stages of the generated Markov processes, until the steady-state is reached. The number of steps until the steady-state behaviour is reached has been estimated through experimentation. Since only DTMCs are considered for simplicity, the probability distribution at different steps has been calculated using simple linear algebra. The results are presented in a logarithmic scale to maximise visibility.



(a) Unstructured DTMCs

(b) Quasi-lumpable DTMCs

(c) Nearly completely decomposable DTMCs

Figure 3.2: For a number of randomly generated DTMCs, we have calculated the K-L divergences (in logarithmic scale) between the true and the approximate state-space distribution at different times, for the two aggregation approaches: NCD-based and quasi-lumpability. Three different classes of DTMCs have been considered: unstructured, quasi-lumpable and nearly-completely decomposable.

We have considered three different classes of randomly generated models. The first class involves models that feature no particular structure. For each DTMC, the entries of the transition probability matrix were set to be non-zero with probability 0.1. The non-zero entries were drawn from a continuous uniform distribution and subsequently normalised. Figure 3.2(a) depicts the results of 10 unstructured DTMCs. It appears that there is a small tendency for the quasi-lumpability approach to produce smaller values for the K-L divergence. We would expect that an approach that relies on quasi-lumpability would be more accurate, as quasi-lumpability represents a more generic family of approximate equivalences. We have to note however that the approach as described in Algorithm 2 is only sub-optimal. Recall that only an upper bound on a metric related to quasi-lumpability is minimised, rather than the metric itself. Given the current state of Algorithm 2, our suspicion is that models should favour one or the other approach, depending on structural properties.

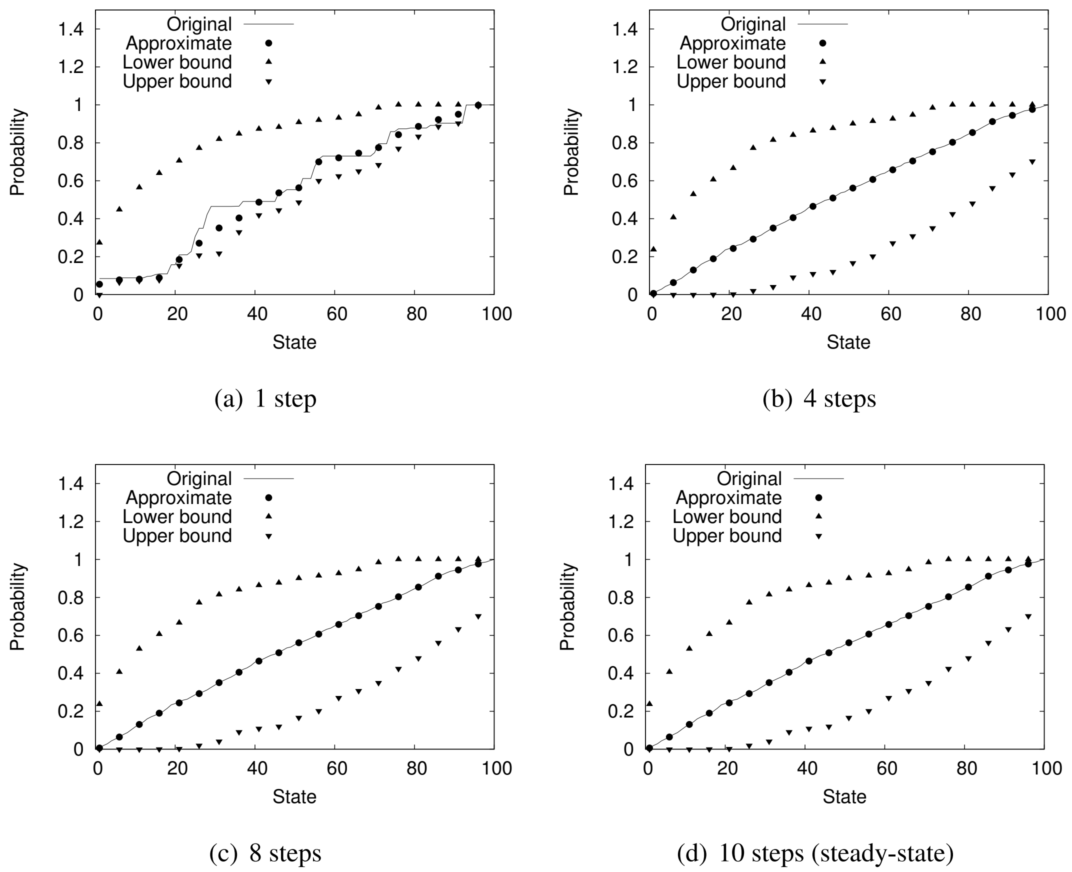
The next two classes of models include DTMCs that have been chosen to be either lumpable or nearly-completely decomposable with respect to some random partition. In this way, we know that there is a behavioural pattern to be discovered. In particular, the second class involves quasi-lumpable models. For each DTMC, we have first generated a lumped version of 100 states in the same way as the unstructured models. Then, a random mapping from 400 states to 100 has been produced, and each transition in the aggregated state-space has been randomly distributed to the states of the corresponding class. Eventually, noise was introduced to those lumpable models, ensuring that they are quasi-lumpable. Figure 3.2(b) depicts the results of 10 quasi-lumpable DTMCs. It appears that the quasi-lumpability approach has a stronger tendency to produce accurate results this time, compared to the NCD-based approach.

The last class involves nearly-completely decomposable models. For each DTMC, 100 strongly connected components have been randomly generated, while weak transitions have been added between components. Figure 3.2(c) depicts the results of 10 nearly-completely decomposable DTMCs. The two aggregation approaches seem to be quite similar in terms of accuracy for many instances. There are some instances however for which the quasi-lumpability approach fails to identify a good partitioning, in contrast with the NCD-based approach which appears to be more robust. As a final comment on the experimental results presented, we think that the applicability of each aggregation approach depends on the properties of the model in question.

3.4.2 Effect on Stochastic Bounds

We have seen that both aggregation approaches that we consider in this chapter produce reasonable approximations for some initial Markov chain, despite the fact that they cannot produce any guarantees for the calculated probabilities in the form of probability bounds. Nevertheless, such probability bounds can be easily obtained by using any relevant method in the literature. As a matter of fact, approximate aggregation as discussed in this thesis results results in a partition of the state-space, which can be used to obtain lumpable stochastic bounds for the original Markov chain.

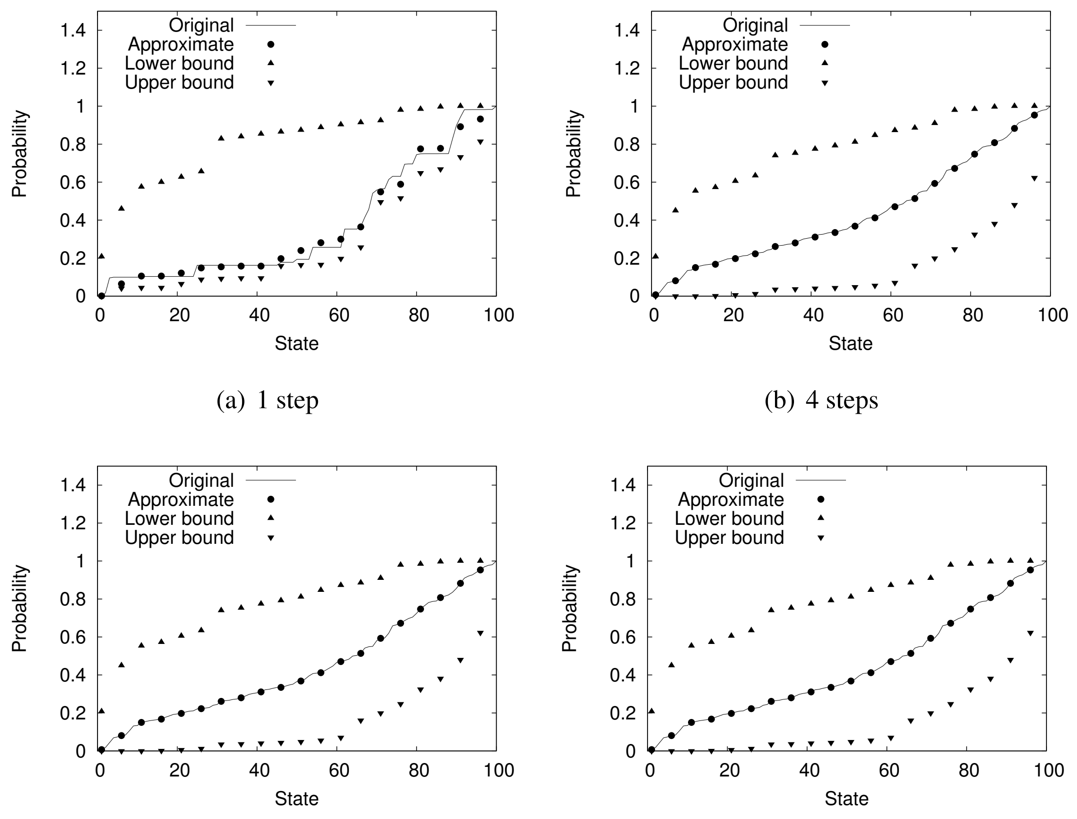
What we hope is that an appropriate partition would probably produce tight bounds. Unfortunately, that does not seem to be the case as we shall see by a counter-example. We consider one of the unstructured randomly generated models of the experiment in Figure 3.2(a). The model in question features 400 states and it has been reduced to 100 states, just as in the experiments in the previous section. We plot the CDFs of an aggregated model $\tilde{P}$ against the CDFs of the initial Markov chain, as long as the upper and lower bounds obtained by the LIMSUB algorithm [33]. Each diagram in the figures that follow depicts these CDFs at a different time. Note that the probabilities presented are in the aggregated state-space.



(a) 1 step (b) 4 steps (c) 8 steps (d) 10 steps (steady-state)

Figure 3.3: CDF approximation at different times for the NCD-based approach. A randomly generated DTMC has been considered.

In Figure 3.3 we present the results for a partition obtained using the NCD-based aggregation approach. Although the CDF approximation appears to be quite accurate, the bounds are not tight enough to provide us any useful information. It appears that there is little if no relation between the approximation quality and the quality of the bounds.



(a) 1 step (b) 4 steps

(c) 8 steps (d) 10 steps (steady-state)

Figure 3.4: CDF approximation at different times for the quasi-lumpability approach. A randomly generated DTMC has been considered.

The first result was rather discouraging, but we shall also examine the effect of a partition given by the quasi-lumpability approach in Figure 3.4. Despite the accurate approximation result, just as happen with the NCD-based approach, the stochastic bounds do not appear to be informative either.

These empirical results only verify that bounding the stochastic behaviour of an aggregated Markov chain is a great challenge. Maybe the biggest issue with stochastic comparison is that it is required that at least one of the two matrices is monotone. The need to make the bounding chain monotone destroys the similarity with the original Markov chain. The closer to monotone the original matrix is, the fewer manipulations will be required to obtain an upper bound. In that case, stochastic bounds could be tight enough to produce useful results.

Nevertheless, our example has shown that even in cases where stochastic comparison of Markov chains cannot produce bounds that are tight enough to be informative, approximate aggregation may still be able to produce estimates of the state probabilities that are accurate enough.

3.5 Summary

The problem of Markov chain aggregation has been traditionally treated in terms of the notion of NCD. We have reviewed several approaches that rely on the spectral properties of transition probability matrices to identify nearly-completely decomposable parts of the state-space. We have seen that spectral segmentation of Markov chains is associated with the fundamental aspects of spectral clustering, and we have adapted a spectral clustering algorithm to perform Markov chain aggregation.

We have also defined an alternative strategy of state-space partitioning that relies on the concept of quasi-lumpability. More specifically, quasi-lumpability has been associated with the minimisation of the $E_{i,j}$ measure between states in the same class. We have shown that a simple clustering algorithm can be used to obtain an upper bound for this measure. Intuitively, the quasi-lumpability approach should be superior, since a nearly completely decomposable system is essentially quasi-lumpable, but not vice-versa. Experimental results do not support this hypothesis though. In fact, it appears that some models favour the quasi-lumpability approach, while others the NCD-based approach. This can be attributed to the fact that the quasi-lumpability method is sub-optimal, since it minimises only an upper bound for $E_{i,j}$. A direct optimisation of $E_{i,j}$ is challenging, since the $E_{i,j}$ measure between states i and j will be different for different partitions of the state-space, we think however it is an interesting subject for future research.

Given a state-space partition by any of the two aforementioned partitioning approaches, we have discussed how an approximation of the original Markov chain can be constructed. The transition probabilities in the aggregated state-space have been summarised by the average transition probabilities from one class to another. We have commented on the use of a weighting scheme, and we have shown that a uniform weighting vector is the only reasonable choice in the general case. Moreover, we have shown that the behaviour of a uniformly lumped Markov chain lies within the lumpable stochastic bounds that can be obtained for a given partition.

Chapter 4

Compositional Aggregation

So far we have considered approximate aggregation for unstructured Markov chains. These methodologies do not rely on particular properties that the modeller must be aware of. The idea was to employ an unsupervised machine learning algorithm that is able to detect behavioural patterns in the state-space that might not be visible from a high-level perspective.

Although aggregation strategies free of assumptions related to structure are definitely desirable, they typically come at a high computational cost. The process of aggregating the Markov chain using such means is easily more expensive than solving it directly. The NCD-based approach is a characteristic example, as it requires the calculation of several of the eigenvectors of the transition probability matrix, whereas only the first eigenvector is required for solving for the steady-state behaviour. Similarly, the quasi-lumpability approach involves the use of a spectral clustering algorithm, which also requires several eigenvectors.

Completely unstructured Markov chains are almost never assumed in modelling. Markov chains are typically generated by formalisms such as queueing networks [57, 89, 74], Petri nets [76], or stochastic process algebras. Such modelling languages provide a high-level representation for a Markovian system. In the case of PEPA in particular, models are described as combinations of components that interact with each other. Although no particular structure is assumed for the components themselves, the existence of components provides a structure that we can rely on to efficiently aggregate the model. A component state-space is significantly smaller than the state-space of the whole system, which is bounded above by the product of all the components involved. An approximation on the component level can be very efficient and it can also result in a considerable reduction of the global state-space.

In this chapter we discuss how approximate aggregation can be applied in a compositional way for PEPA models. PEPA components can be mapped to labelled transition systems, whose transitions feature exponentially distributed delays. In other words, PEPA components are essentially labelled CTMCs, a fact that makes all of the discussions on approximate CTMC aggregation relevant.

In Section 4.1 we establish the conventions followed throughout this chapter. In Section 4.2 we develop an appropriate notion of approximate component equivalence. The issues related to approximate component aggregation are described in Section 4.3. Section 4.4 investigates composition for aggregated components. We follow two distinct approaches to this problem. The first solution involves a Kronecker representation, which is a refinement of the method we have used in [72]. In order to deal with some limitations of the Kronecker representation, we alternatively propose a new structured operational semantics for the PEPA language that takes into account the aggregated component state-space. Finally, examples of the compositional aggregation approaches are given in Section 4.5.

4.1 Terminology and Conventions

In the PEPA language, a component name C is bound to a process definition. According to the semantics of the language, the derivative set of C, denoted as $ds(C)$, along with the activities involved define a labelled transition system. In terms of the current chapter, a component C will actually refer to the labelled transition system with state-space $ds(C)$, and transitions which are specified by the semantics of Figure 2.1. In other words, our use of the term "component" involves the set of process definitions for which the semantics induce a directed graph with one connected component.

According to standard PEPA notation [48], the set of actions of which a component C is capable is denoted by $\mathcal{A}(C)$. Since we are interested in the labelled transition system induced by C, we shall make use of the *complete set of actions* of a component C, which is defined as follows:

$$\vec{\mathcal{A}}(C) = \bigcup_{C' \in ds(C)} \mathcal{A}(C') \tag{4.1}$$

Assuming an ordering of states in the derivative set $ds(C)$, we shall describe the behaviour of components in terms of matrices, whose entries contain the rates of each individual activity. Given the complete set of actions of C, we have an *action rate* *matrix* R_a for every action $a \in \vec{\mathcal{A}}(C)$. The entries of each R_a matrix are determined by the language semantics:

- An entry $R_{a,ij} > 0$ will denote the rate of the activities of type a from the i-th state to the j-th state of C.
- A value 0 for $R_{a,ij}$ simply means that there is no activity of type a from the i-th to the j-th state of C.
- An unspecified rate $w \times \top$ for $R_{a,ij}$ implies that the activities of type a from the i-th to the j-th state of C are passive, having weight $w \in \mathbb{N}$.

If we ignore the action labels, we can simply calculate the sum $R = \sum_{a \in \vec{\mathcal{A}}(C)} R_{a,ij}$, which is the transition rate matrix of the underlying CTMC. A partition on the state-space of C can be obtained if we apply any of the partitioning approaches discussed in Chapter 3; these are the NCD-based approach in Section 3.1, and the quasi-lumpability approach of Section 3.2. In fact, R cannot be calculated at all times, as some of the activities are shared, meaning that the corresponding rates depend on other components. In the section that follows we explain which of the action rate matrices are used as input for the partitioning process.

Any connected component prior to the application of the cooperation operator constitutes the labelled transition system of a sequential component. By the term "parallel component" we refer to the connected labelled transition system that is derived by the application of the cooperation operator for any two connected components. Parallel and sequential components are not substantially different; as it has been shown in [24] for every parallel component there exists a sequential one that is isomorphic. In the experiments of Section 4.5, we apply the approximate reduction algorithms to populations of identical components. For the sake of simplicity however, in most of the discussion in this chapter we consider small sequential components.

4.2 On Equivalence of PEPA Components

Compositional approximation requires that we replace C with a component $\tilde{C}$ that is approximately equivalent, where $\tilde{C}$ results from aggregating C. Equivalence of Markov chains of different size is adequately captured by the notion of lumpability. By relaxing the conditions imposed by lumpability, it was possible to define an approximate version called “quasi-lumpability”. We intend to apply a similar approach for the state-space of PEPA components. In this section, we investigate different notions of PEPA component equivalence and we discuss how these can be applied in an approximate setting.

4.2.1 Strongly Equivalent Components

In the original work on PEPA [48], several equivalences for PEPA components have been introduced, including isomorphism, weak isomorphism, strong bisimilarity and strong equivalence. Isomorphism is not really suitable for our purposes here, as the components involved are required to have state-spaces of the same size. Weak isomorphism and strong bisimilarity allow components of different size, however they do not induce lumpability for the underlying CTMCs.

Strong equivalence combines the two desirable features needed to define approximate component equivalence in the style of quasi-lumpability. It has been shown in [48] that if two components C and $\tilde{C}$ are strongly equivalent then their underlying CTMCs will be lumpably equivalent. Moreover, the compositions $C \underset{L}{\bowtie} C'$ and $\tilde{C} \underset{L}{\bowtie} C'$ will be also strongly equivalent for any C'.

In the context of state-space aggregation, a component $\tilde{C}$, which is strongly equivalent to C, results from aggregation given some partition on the state-space of C. Strong equivalence can then be defined in terms of the action rate matrices and a partition Δ on the state-space of C.

Definition 10 (Strong Equivalence)**.** *Let C be a PEPA component with transition rate matrices R_a, $\forall a \in \vec{\mathcal{A}}(C)$. A partition $\Delta = \{A_1, \ldots, A_K\}$ on $ds(C)$ induces a strongly equivalent aggregated component, if for any two classes $A_k, A_l \in \Delta$, and for any two states $i, j \in A_k$:*

$$\sum_{m \in A_l} R_{a,im} = \sum_{m \in A_l} R_{a,jm}, \quad \forall a \in \vec{\mathcal{A}}(C) \tag{4.2}$$

Although strong equivalence for any PEPA component induces a lumpable partition for the underlying Markov chain, we can achieve the same result using some weaker notion of equivalence. In fact, we can easily see that strong equivalence as defined in [48] is too strict, especially in an approximate setting. For example, let us consider the following component definition:

$$\begin{aligned}
P_1 &\overset{def}{=} (a,r).P_2 + (a,r).P_3 \\
P_2 &\overset{def}{=} (b,r).P_1 + (b,r).P_3 \\
P_3 &\overset{def}{=} (c,r_c).P_1
\end{aligned}$$

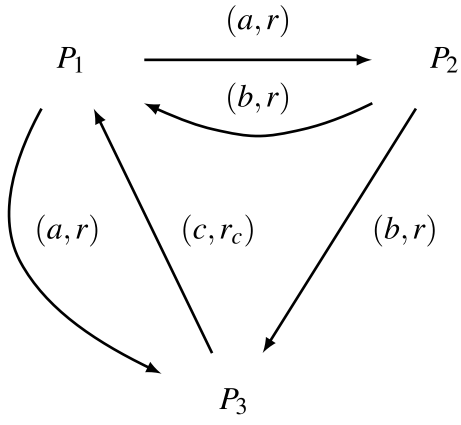


Figure 4.1: Derivation graph for P_1

By applying the PEPA operational semantics, we can obtain the derivation graph of Figure 4.1, which is characterised by the following rate matrices:

$$R_a = \begin{bmatrix} 0 & r & r \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad R_b = \begin{bmatrix} 0 & 0 & 0 \\ r & 0 & r \\ 0 & 0 & 0 \end{bmatrix}, \quad R_c = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ r_c & 0 & 0 \end{bmatrix}$$

The underlying CTMC is lumpable to the partition $\Delta = \{\{P_1, P_2\}, \{P_3\}\}$. Lumpability follows from the fact that both P_1 and P_2 may perform a transition to P_3 at the same rate, as we can see in the transition rate matrix $R = R_a + R_b + R_c$:

$$R = \begin{bmatrix} 0 & r & r \\ r & 0 & r \\ r_c & 0 & 0 \end{bmatrix}$$

Nevertheless, P_1 and P_2 are not strongly equivalent, since P_3 is reached by activities of different action type in each case. If there is no synchronisation on either action a or b, we could treat them as a single action type and therefore obtain a lumpable partition. This is an example of strong equivalence being too strict, as there would be no problem to characterise $\{P_1, P_2\}$ as a set of equivalent components.

In an approximate aggregation setting, strong equivalence might also be problematic for one more reason. In Chapter 3 we have discussed some approaches to find a nearly optimal partition for a Markov chain. A key characteristic of these approaches is that they rely heavily on data. Strong equivalence is judged by several transition rate matrices, one for each action. In contrast, lumpability is judged by a single matrix which holds the sums of the individual activity rates. Lumpability depends on a structure that is typically more dense. We need a form of equivalence that imposes identical behaviour for shared actions, while the individual component actions are treated as a single action type, just as in the case of lumpability.

4.2.2 Modified Strong Equivalence

A weaker form of strong equivalence has been defined for stochastic automata in [2], which has been used in terms of an approximate aggregation process in [3]. In order to adopt this definition in the context of PEPA models, it is helpful to consider the following lemma:

Lemma 1. *Let C be a PEPA component with $\vec{\mathcal{A}}^-(C)$ set of individual actions and $\vec{\mathcal{A}}^+(C)$ set of shared actions, where $\vec{\mathcal{A}}^-(C) \cup \vec{\mathcal{A}}^+(C) = \vec{\mathcal{A}}(C)$ and $\vec{\mathcal{A}}^-(C) \cap \vec{\mathcal{A}}^+(C) = \emptyset$. We shall say that the* individual behaviour *of C is characterised by the* individual rate matrix*:*

$$R^- = \sum_{a \in \vec{\mathcal{A}}^-(C)} R_a \tag{4.3}$$

while the shared behaviour *of C is characterised by the rate matrices: R_a, $\forall a \in \vec{\mathcal{A}}^+(C)$.*

A useful observation regarding Lemma 1 is that we can summarise the individual behaviour of any component in a single rate matrix. We shall now define *modified strong equivalence* for a component C in terms of a partition on its state-space:

Definition 11 (Modified Strong Equivalence)**.** *Let C be a PEPA component with transition rate matrices R_a, $\forall a \in \vec{\mathcal{A}}(C)$. A partition $\Delta = \{A_1, \ldots, A_K\}$ on $ds(C)$ induces an equivalent aggregated component in the sense of modified strong equivalence, if for any two classes $A_k, A_l \in \Delta$, and for any two states $i, j \in A_k$:*

$$\sum_{m \in A_l} R_{a,im} = \sum_{m \in A_l} R_{a,jm}, \quad \forall a \in \vec{\mathcal{A}}^+(C) \tag{4.4}$$

and

$$\sum_{m \in A_l} R^-_{im} = \sum_{m \in A_l} R^-_{jm} \tag{4.5}$$

In the case of modified strong equivalence, two components are considered to be equivalent if they agree on the shared actions and on the total of individual actions only. As long as an activity is local, we do not really care what the action type is in the context of modified strong equivalence. For the shared actions however, there is no legitimate way to produce a meaningful summary in a single matrix. Recall that the rates of these actions are not fully determined in isolation, as they depend on the global state. More specifically, the PEPA semantics impose that the rate of a shared activity is the minimum of the apparent rates of the components involved. This minimum also depends on the state of the other cooperating component, so it cannot be determined *a priori* i.e. without deriving the global state-space.

By relaxing the conditions of modified strong equivalence in a way similar to quasi-lumpability, we obtain the following approximate notion of equivalence for PEPA components:

Definition 12 (Modified Quasi-Strong Equivalence)**.** *Let C be a PEPA component with transition rate matrices R_a, $\forall a \in \vec{\mathcal{A}}(C)$. A partition $\Delta = \{A_1, \ldots, A_K\}$ on $ds(C)$ induces an approximately equivalent aggregated component in the sense of modified quasi-strong equivalence, if for any two classes $A_k, A_l \in \Delta$, and for any two states $i, j \in A_k$:*

$$\left| \sum_{m \in A_l} R_{a,im} - \sum_{m \in A_l} R_{a,jm} \right| \leq \varepsilon, \quad \varepsilon \geq 0, \quad \forall a \in \vec{\mathcal{A}}^{+}(C) \tag{4.6}$$

and

$$\left| \sum_{m \in A_l} R^{-}_{im} - \sum_{m \in A_l} R^{-}_{jm} \right| \leq \varepsilon, \quad \varepsilon \geq 0 \tag{4.7}$$

According to the PEPA semantics, the individual activities of a component C induce a CTMC, whose generator matrix can be directly derived by appropriately modifying the diagonal entries of the individual rate matrix R^{-}. Note that this CTMC does not accurately capture the behaviour of C, as the shared actions have been ignored. However, if the set of shared actions is relatively small, we can expect that R^{-} will be much more dense than the total of R_a, $\forall a \in \vec{\mathcal{A}}^{+}(C)$. If this condition holds and R^{-} captures a significant part of the component behaviour, it should be reasonable to apply an approximate aggregation algorithm to R^{-}, in order to obtain a nearly optimal partition Δ with respect to the individual behaviour of C.

4.2.2.1 Dealing with Deadlocks

Finally, we discuss a practical issue with respect to the use of the individual rate matrix R^{-} as input to the partitioning approaches of Chapter 3. In practice, ignoring a shared activity in a particular component may introduce deadlocks in its behaviour.

For example, consider a component C with a shared action type a and the following rate matrices:

$$R^- = \begin{bmatrix} 0 & 0 & 0 & 0 & 2 \\ 3 & 0 & 6 & 0 & 0 \\ 0 & 3 & 0 & 0 & 4 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 5 & 0 \end{bmatrix} \qquad R_a = \begin{bmatrix} 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 3 & 0 & 6 & 0 \\ 0 & 0 & 2 & 0 & 0 \end{bmatrix}$$

In the example above, R^- contains a deadlock at the fourth state, meaning that there is no non-trivial steady-state distribution over R^- in isolation, hence no way to compute the reversible process needed to apply the NCD-based approach, as described in Sect. 3.1.2.2. To solve this problem, we use the $\hat{R}^-$ matrix instead, which is constructed as in the following example:

$$\hat{R}^- = \begin{bmatrix} 0 & 0 & \varepsilon & 0 & 2 \\ 3 & 0 & 6 & 0 & 0 \\ 0 & 3 & 0 & \varepsilon & 4 \\ 0 & \varepsilon & 0 & \varepsilon & 0 \\ 0 & 0 & \varepsilon & 5 & 0 \end{bmatrix}$$

where $\varepsilon > 0$ is a small rate added to some transition for each shared activity. Hence, if the original PEPA model contains no deadlocks, we can be sure that $\hat{R}^-$ will have no deadlocks either. By doing so, we obtain a partition of the component's state-space by using only a part of its behaviour. The ε rates added are equally distributed and therefore imply ignorance about the shared activity rates.

4.3 Approximate Aggregation of Components

We assume some component C and a partition Δ over its state-space, which has been produced after applying a partitioning strategy to the corresponding individual rate matrix R^-. The next step is to derive the activities of the aggregated $\tilde{C}$ component, whose states are labelled by the instances of Δ.

4.3.1 Real Rate Matrices

The transitions of any component C can be described by a set of rate matrices $\{R_a : a \in \vec{\mathcal{A}}(C)\}$, which correspond to the action types that C is capable of. Let us first consider the case where no passive activities are involved, so for any action type a we have $R_a \in \mathbb{R}^{N\times N}$, where N is the number of states in $ds(C)$. In Equation (3.24) of Section 3.3, we have defined uniform lumping as the process of constructing a reduced matrix by averaging the transitions to the other classes. We can apply a similar lumping process to each one of the rate matrices individually. More formally, given a component C and a partition $\Delta = \{A_1, \ldots, A_K\}$ on $ds(C)$ with K classes, then for each instance of $\{\tilde{R}_a : a \in \vec{\mathcal{A}}(C)\}$ we have the following $K \times K$ rate matrix:

$$\tilde{R}_{a,ij} = \frac{\sum_{k \in A_i} \sum_{l \in A_j} R_{a,kl}}{|A_i|} \tag{4.8}$$

where $|A_i|$ denotes the number of states included in class A_i.

A PEPA component is approximately aggregated by aggregating all of its rate matrices according to Equation (4.8), in the style of uniform lumping of Markov chains in Chapter 3. The newly produced collection $\{\tilde{R}_a : a \in \vec{\mathcal{A}}(C)\}$ fully describes the behaviour of the aggregated component $\tilde{C}$, whose state-space is marked by Δ. It can be easily shown that the sum of the aggregated action matrices $\sum_{a \in \vec{\mathcal{A}}(C)} \tilde{R}_a$ is equal to the aggregation of the sum of the original rate matrices, since the order of summation does not matter.

The possibility of weighted lumping with respect to a vector $\boldsymbol{w}$ has already been investigated for probability matrices in Section 3.3.1. According to Theorem 2, the optimal choice for $\boldsymbol{w}$ depends of the current state probabilities. Assuming that a Markov chain is not in steady-state, its state probabilities change over time, and therefore there is no globally optimal choice for $\boldsymbol{w}$. We think that uniform lumping is appropriate, as it captures our uncertainty regarding the state distribution. This result can be easily extended for rate matrices via uniformisation.

4.3.2 Rate Matrices with Unspecified Rates

In this section, we consider the case where a rate matrix describes passive activities. More specifically, we assume that the entries of a rate matrix are exclusively either unspecified or zeros, that is $R_a \in \{0, \top\}^{N \times N}$. It is possible to apply (4.8) in order to get an approximately lumped version of it.

Recall that an activity with rate $\top$ may be assigned a weight $w \in \mathbb{N}$, which determines the relative probability of that particular activity, while the absence of a weight simply implies that $w = 1$. An unspecified rate $\top$ should not be interpreted as “infinity”, as it is important to keep track of its probability. Therefore, the process of aggregating a rate matrix $R_a \in \{0, \top\}^{N \times N}$ is similar to the process of aggregating a probability matrix, as seen in Chapter 3. The effect of approximate aggregation will be on the weights that are associated with any passive rate $\top$, rather than on $\top$ itself.

Approximate aggregation as formulated in Equation (4.8) involves two summations: given a state indexed by k we calculate the total rate of transitioning to a class A_j, and we average these total rates for every $k \in A_i$. The addition of two unspecified rates $w_1 \times \top$ and $w_2 \times \top$ is defined in [48] as follows:

$$w_1 \times \top + w_2 \times \top = (w_1 + w_2) \times \top \qquad \forall w_1, w_2 \in \mathbb{N} \tag{4.9}$$

However, in (4.8) we also have to divide the sum of the rates by the size of the outgoing class. Although division of an unspecified rate with a real number is not originally defined in [48], there is no reason why we cannot have such an operation. In fact, Smith [90] has defined multiplication of $\top$ by a real number c. Following Smith, we can more conveniently write:

$$\frac{w \times \top}{c} = \frac{w}{c} \times \top \tag{4.10}$$

If weights can be written as fractions as in (4.10), then the weight of an aggregated component could be a real rather than a natural number as required in [48]. Nevertheless, there should be no technical problem with considering weights that are positive real numbers. Regarding the behaviour of $\top$ in the context of the minimum operator, that will remain unchanged:

$$\min(w_1 \times \top, r) = \begin{cases} r, & r \in \mathbb{R}^+ \\ \min(w_1, w_2) \times \top, & r = w_2 \times \top \end{cases} \tag{4.11}$$

where $w_1, w_2 \in \mathbb{R}^+$.

In an approximate aggregation context, we allow w to be any positive real number. It can be easily seen that w will never be negative or zero; Equation (4.8) dictates that a weight w for an aggregated component will be the average of a set of values which are assumed to be positive.

4.3.3 Rate Matrices with both Active and Passive Activities

In this section, we demonstrate that rate matrices that contain a mixture of real-valued and unspecified rates are problematic. However, it is possible to avoid such an occurrence, if we impose certain restrictions on the model definition.

According the original work on PEPA [48], a component C is not allowed to have an action type a that involves both active and passive activities. This requirement follows directly from the fact that the apparent rate $r_a(C)$ cannot be defined, since the addition of an active rate $r \in \mathbb{R}^+$ with an unspecified rate $\top$ is not defined as an operation in [48]. Recall that the apparent rate $r_a(C)$ is the sum of all of the rates of the activities of type a in $\mathcal{A}ct(C)$. Therefore, it is important to impose that all of the activities of type a are consistently active or passive for the same C component. Note that the term "component" in the language definition refers to a state rather than a labelled transition system.

The following is an example of a valid PEPA component, i.e. an apparent rate can be defined for each one of its states.

$$\begin{aligned} C &\overset{def}{=} (a, r_1).C_1 + (a, r_2).C_2 \\ C_1 &\overset{def}{=} (b, r_3).C \\ C_2 &\overset{def}{=} (b, \top).C \end{aligned}$$

In the context of aggregation however, C can be problematic, as it might give rise to an invalid aggregated component. For C we have actions a and b, each of which corresponds to a rate matrix:

$$R_a = \begin{bmatrix} 0 & r_1 & r_2 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \qquad R_b = \begin{bmatrix} 0 & 0 & 0 \\ r_3 & 0 & 0 \\ \top & 0 & 0 \end{bmatrix}$$

Assuming a partition of the state-space $\Delta = \{\{C\}, \{C_1, C_2\}\}$, we can aggregate both matrices using Equation (4.8) to obtain:

$$\tilde{R}_a = \begin{bmatrix} 0 & r_1 + r_2 \\ 0 & 0 \end{bmatrix}, \qquad \tilde{R}_b = \begin{bmatrix} 0 & 0 \\ 0.5 \times r_3 + 0.5 \times \top & 0 \end{bmatrix}$$

The aggregated rate matrix $\tilde{R}_b$ is not valid, as there is no way to calculate the sum of the real rate $0.5 \times r_3$ with the unspecified rate $0.5 \times \top$. For the same reason, the apparent rate for action b cannot be defined. Therefore, the aggregated component is not a valid PEPA component, according to the original work on PEPA [48].

In order to overcome this problem, we have to impose an additional restriction. That is that for any PEPA component C, it should not allowed to have action types that involve both active and passive activities for the derivative set $ds(C)$. The C component of the previous example is not valid according to this specification. We can however rename the different occurrences of the b action type in such a way that we obtain the following valid component definitions:

$$\begin{aligned} C &\overset{def}{=} (a, r_1).C_1 + (a, r_2).C_2 \\ C_1 &\overset{def}{=} (b_1, r_3).C \\ C_2 &\overset{def}{=} (b_2, \top).C \end{aligned}$$

In the new version of C, there is consistency of active and passive actions over all of the states of the transition system. This is reflected in the following rate matrices that correspond to the actions that C, C_1 and C_2 are able to perform:

$$R_a = \begin{bmatrix} 0 & r_1 & r_2 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \qquad R_{b_1} = \begin{bmatrix} 0 & 0 & 0 \\ r_3 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \qquad R_{b_2} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ \top & 0 & 0 \end{bmatrix}$$

Given a partition $\Delta = \{\{C\}, \{C_1, C_2\}\}$, we can aggregate all three matrices using (4.8) to obtain:

$$\tilde{R}_a = \begin{bmatrix} 0 & r_1 + r_2 \\ 0 & 0 \end{bmatrix}, \qquad \tilde{R}_{b_1} = \begin{bmatrix} 0 & 0 \\ 0.5 \times r_3 & 0 \end{bmatrix}, \qquad \tilde{R}_{b_2} = \begin{bmatrix} 0 & 0 \\ 0.5 \times \top & 0 \end{bmatrix}$$

To conclude, the rate matrix of a valid component C for some action a can be either $R_a \in \mathbb{R}^{N \times N}$ or $R_a \in \{0, \top\}^{N \times N}$. In both cases, Equation (4.8) will be used to produce the aggregated form for the transitions of type a.

4.4 Semantics for Aggregated PEPA models

Our objective is to aggregate PEPA components individually and reflect this reduction on the underlying CTMC. Therefore, we need a compositional representation of the corresponding generator matrix. In [72], we have used a Kronecker representation of PEPA models, where the “global” generator matrix is expressed in terms of “partial” generator matrices combined via Kronecker algebra. We have successfully produced reduced versions of such partial generator matrices, which were then combined to obtain an approximately aggregated state-space for the entire model.

In this section, we discuss this Kronecker algebra approach in more detail, and we explore certain aspects that have not been investigated in [72]. As we shall see, one limitation is the fact that the generator matrix produced might include states that are not reachable in the derivation graph. In order to overcome this issue, we additionally define a structured operational semantics characterisation of the cooperation of approximately aggregated components.

4.4.1 The Kronecker Algebra Approach

Kronecker representation for PEPA models has been originally proposed in [50]. In this section, we propose an alternative representation based on Kronecker algebra that is more appropriate for approximate aggregation of PEPA components. The following Kronecker operators are used by both approaches:

Definition 13 (Kronecker Product)**.** *Given a $m \times n$ matrix A and $p \times q$ matrix B, their* Kronecker product *$C = A \otimes B$ is a $mp \times nq$ matrix with elements:*

$$\begin{gathered} C_{ab} = A_{ij} B_{kl} \\ \text{where} \quad a = p(i-1)+k \quad \text{and} \quad b = q(j-1)+l \end{gathered} \tag{4.12}$$

Definition 14 (Kronecker Sum)**.** *Given a $n \times n$ matrix A and $m \times m$ matrix B, their* Kronecker sum *is defined as follows:*

$$A \oplus B = A \otimes I_m + I_n \otimes B \tag{4.13}$$

where I_n is the $n \times n$ identity matrix, and I_m is the $m \times m$ identity matrix.

As shown in [50], the generator matrix Q that corresponds to a PEPA model can be represented as a Kronecker product of terms in the following way:

$$Q = \bigoplus_{i=1}^{N} R_i + \sum_{a \in \mathcal{A}} r_a \times \left(\bigotimes_{i=1}^{N} P_{i,a} - \bigotimes_{i=1}^{N} \bar{P}_{i,a} \right) \tag{4.14}$$

where

- N is the number of components that appear in the system equation. The subscript i corresponds to a particular component C_i.
- $\mathcal{A}$ is the set of shared actions.
- R_i is the rate matrix of i-th component based on its individual actions:

$$R_i = \sum_{a \in \vec{\mathcal{A}}(C_i) \setminus \mathcal{A}} R_{i,a} \tag{4.15}$$

- r_a is the minimum *functional rate* of the shared action a over all components. The term "functional rate" implies that the rate of an action depends on the state of one or more components. Equivalently, there is a single rate function $r_a(C)$ that describes the apparent rate of action a for each state of component C. The minimum of the functional rates over all components C_i, $i = 1 \dots N$ is defined as follows:

$$r_a = \min(r_a(C_1), r_a(C_2), \dots r_a(C_N)) \tag{4.16}$$

- $P_{i,a}$ is the probability matrix of the i-th component for the shared action a. $\bar{P}_{i,a}$ is a diagonal matrix that ensures that the row sums of the corresponding probability matrix are zero, i.e. it is a valid generator matrix.

Note that $\mathcal{A}$ denotes the set of all shared actions over all components in the model. If some particular component C_i does not perform some action of type a, then this will be reflected on the $P_{i,a}$ matrix, which will essentially be the identity matrix. This implies that the state of C_i is unaffected by action a.

If we observe the summation term in Equation (4.14), we see that r_a, which is the minimum of the functional rates for some action a, is multiplied by the Kronecker product of the $P_{i,a}$ matrices. However, if we aggregate the R_i and $P_{i,a}$ matrices for some component, it is not so clear how the functional rates should be multiplied with the aggregated versions of $P_{i,a}$. It appears that we have to apply the same kind of reduction to the functional rates of individual components, and then combine these functional rates in a way that resembles the Kronecker operators. In order to make this effect on the functional rates explicit, we introduce a new Kronecker representation of PEPA models that is more appropriate for compositional aggregation.

4.4.1.1 Kronecker Representation for Aggregated Components

In order to introduce our Kronecker form of PEPA models, we have to redefine some concepts used in the original approach of Hillston & Kloul [50]. More specifically, Hillston & Kloul have defined a functional rate $r_a(C_i)$ for some action a, as a function from the space of PEPA process definitions to $\mathbb{R}^+ \cup \top$. The value of $r_a(C_i)$ denotes the apparent rate of the process C_i for the action of type a. We shall express the concept of functional rates of components in a vector form instead.

Definition 15 (Apparent Rate Matrix)**.** *Let C_i be a PEPA component whose state-space is indexed by $ds(C_i)$. Let $D_{i,a}$ be a diagonal matrix, also indexed by $ds(C_i)$, whose main diagonal contains the apparent rates of the states of C_i for action a. We define $D_{i,a}$ to be the* apparent rate matrix *of C_i for action a.*

There is one-to-one correspondence between the rows of $D_{i,a}$ and the rows of the probability matrix $P_{i,a}$. We can now express the multiplication with the functional rates purely in terms of linear algebra. The rate matrix of some C_i component for some action a can be written as the following:

$$R_{i,a} = D_{i,a}P_{i,a} \tag{4.17}$$

It can be easily seen that $R_{i,a}$ is exactly the same rate matrix that we would get by multiplying the functional rates $r_a(C_i)$ with $P_{i,a}$. However, we cannot just take the Kronecker product of such rate matrices, as that would not be compatible with the PEPA semantics, which require us to calculate the minimum rather than the product of component rates. In order to combine two rate vectors in a way that is meaningful to these semantics, we have to define a new kind of Kronecker operator.

Definition 16 (Kronecker Minimum)**.** *Given a $m \times n$ matrix A and $p \times q$ matrix B, we define their* Kronecker minimum $C = A \circledcirc_{\min} B$ *to be a $mp \times nq$ matrix with elements:*

$$C_{ab} = \min(A_{ij}, B_{kl}) \\ \textit{where} \quad a = p(i-1)+k \quad \textit{and} \quad b = q(j-1)+l \tag{4.18}$$

We have defined Kronecker minimum in terms of matrices, in the style of the Kronecker product definition. It is now straightforward to construct a diagonal matrix $D_{ii',a} = D_{i,a} \circledcirc_{\min} D_{i',a}$. Therefore, we can reform the Kronecker representation of PEPA in a way that it is purely expressed in terms of linear and Kronecker algebra in the following way:

$$Q = \bigoplus_{i=1}^{N} R_i + \sum_{a \in \mathcal{A}} \underset{i=1}{\overset{N}{\circledcirc_{\min}}} D_{i,a} \cdot \left(\bigotimes_{i=1}^{N} P_{i,a} - \bigotimes_{i=1}^{N} \bar{P}_{i,a} \right) \tag{4.19}$$

To summarise, each component C_i is fully characterised by a collection of matrices: R_i which contains the rates of its individual actions, and for each shared action a we have a probability matrix $P_{i,a}$ and a diagonal matrix $D_{i,a}$ which contains the apparent rates that correspond to the rows of $P_{i,a}$.

One issue that we also have to consider is the case of passive actions, whose rates are set to be $\top$. Equation (4.11) defines the behaviour of $\top$ in the context of the minimum operator. Therefore, the behaviour of $\top$ is also well-defined in terms of the Kronecker minimum. The Kronecker minimum of two or more matrices will only involve a $\top$ if all of the components cooperate on some action, while all of them are passive at the same time. In such a case, the model definition is problematic as it contains a deadlock. Since deadlocks can be identified at the syntax level, it is fair to assume that the model is free of them.

4.4.1.2 Aggregating Kronecker Terms

In the Kronecker representation of Equation (4.19), all the component terms are written as matrices. In short, we have to apply the same partition to all the rate and probability matrices that characterise a component; these are the individual rate matrix R_i, and for each shared action a, the probability matrix $P_{i,a}$ and the apparent rate matrix $D_{i,a}$. More specifically, for any component C_i to be aggregated by a partition Δ_i, we have to apply the following steps:

1. Aggregate R_i to $\tilde{R}_i$, according to Equation (4.8).
2. For each shared action $a \in \mathcal{A}$:
   (a) Calculate $R_{i,a} = D_{i,a}P_{i,a}$.
   (b) Aggregate $R_{i,a}$ to $\tilde{R}_{i,a}$, according to (4.8).
   (c) Aggregate $D_{i,a}$ to $\tilde{D}_{i,a}$, according to (4.8). It is trivial to show that the diagonal matrix $\tilde{D}_{i,a}$ has entries which are the row sums of $\tilde{R}_{i,a}$.
   (d) Calculate $\tilde{P}_{i,a} = \tilde{D}_{i,a}^{-1}\tilde{R}_{i,a}$

There are two possibilities for $R_{i,a}$ and $D_{i,a}$: their entries will be either all positive real numbers, meaning that the associated activities are active, or unspecified if the activities are passive. In both cases there should be no problem with using (4.8) to produce the aggregated versions $\tilde{R}_{i,a}$ and $\tilde{D}_{i,a}$. Regarding Step 2d, it is a simple normalisation of the rate matrix $\tilde{R}_{i,a}$. Note that if $\tilde{R}_{i,a}$ and $\tilde{D}_{i,a}$ contain unspecified rates, the matrix inversion $\tilde{D}_{i,a}^{-1}$ is not normally defined. By convention, we simply replace any unspecified rate $\top$ with "1" in both matrices, but we keep their weights, as they represent the relative probabilities for the corresponding activities. In this way, we produce a real-valued matrix $\tilde{P}_{i,a}$ that captures the activity probabilities for the aggregated component.

The computation of $\tilde{P}_{i,a}$, which is the aggregated probability matrix of the i-th component for action a, might seem to be more complicated than it is required to be. One could argue that we could directly aggregate $P_{i,a}$ to $\tilde{P}_{i,a}$, using Equation (4.8). This approach would be problematic however, since any rate information is removed from $P_{i,a}$, so the probability mass is not distributed in the same manner as it would be in the case of the rate matrix $R_{i,a}$. Thus, the normalisation and the subsequent aggregation of $R_{i,a}$ is necessary to describe the correct aggregated behaviour.

4.4.1.3 Limitations of Kronecker Representation

In this section, we highlight an intrinsic limitation of the Kronecker representation of PEPA models, in that unreachable states might be included in the resulting generator matrix. In the general case, the state-space of any parallel composition of components is bounded by the Cartesian product of the individual state-spaces involved. That is the state-space of such a composition cannot be larger than the total of the tuples that denote the possible state combinations for the components in the model. For example, an empty cooperation of two components implies that any of the two may perform any transition independently, therefore any ordered pair that characterises the combined state is reachable in the derivation graph.

Many times however, the state-space of a PEPA cooperation component is only a subset of the Cartesian product, as some of the states in the Cartesian product space are not reachable in the derivation graph produced by the language semantics (Figure 2.1). For example, let us consider the composition $P_1 \underset{\{a,b\}}{\bowtie} Q_1$ of the following components:

$$\begin{aligned}
P_1 &\overset{def}{=} (c,r).P_2 + (a,\top).P_3 & \qquad Q_1 &\overset{def}{=} (a,r_a).Q_2 \\
P_2 &\overset{def}{=} (c,r).P_1 & Q_2 &\overset{def}{=} (b,r_b).Q_1 \\
P_3 &\overset{def}{=} (c,r).P_4 \\
P_4 &\overset{def}{=} (c,r).P_3 + (b,\top).P_2;
\end{aligned}$$

$$\begin{tikzcd}[row sep=huge, column sep=huge]
P_1 \underset{\{a,b\}}{\bowtie} Q_1 \arrow[r, "(a,r_a)"] \arrow[d, swap, "(c,r)"] & P_3 \underset{\{a,b\}}{\bowtie} Q_2 \arrow[d, swap, "(c,r)"] \\
P_2 \underset{\{a,b\}}{\bowtie} Q_1 \arrow[u, bend right, swap, "(c,r)"] & P_4 \underset{\{a,b\}}{\bowtie} Q_2 \arrow[u, bend right, swap, "(c,r)"] \arrow[l, "(b,r_b)"]
\end{tikzcd}$$

Figure 4.2: Derivation graph for $P_1 \underset{\{a,b\}}{\bowtie} Q_1$

Figure 4.2 depicts the state-space of this composition, where we see that not all of the state combinations are part of the derivation graph. Synchronisation on certain action types implies that certain activities may or may not be enabled depending on the state of the cooperating components.

However, the state-space of the Kronecker representation of a PEPA model is not equal to the size of the derivation graph. Since the cooperation operator of the PEPA language is expressed in terms of Kronecker operations, we know that the dimension of the resulting rate and probability matrices will be $\prod_{i=1}^{N} |C_i| \times \prod_{i=1}^{N} |C_i|$, where $|C_i|$ denotes the size of the C_i component. In practice, that would mean that the composition $P_1 \underset{\{a,b\}}{\bowtie} Q_1$ of our example would involve $4 \times 2 = 8$ states. This is actually an intrinsic limitation of the Kronecker representation which might also affect approximate aggregation. The effects of approximate aggregation might be nullified if the derivation graph is significantly smaller than the full Cartesian product space.

4.4.1.4 A Worked Example

We shall now illustrate the concepts discussed in this section with an example. Consider the composition $P_1 \underset{\{a,b\}}{\bowtie} Q_1$ whose state-space is illustrated in Figure 4.2. We assume a partition $\Delta = \{\{P_1, P_2\}, \{P_3, P_4\}\}$ on the state-space of P_1.

Step 1: Aggregate Components For the P_1 component, we have the following rate matrices that describe its behaviour:

$$\underbrace{\begin{bmatrix} 0 & 0 & \top & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}}_{\text{action } a} \quad \underbrace{\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & \top & 0 & 0 \end{bmatrix}}_{\text{action } b} \quad \underbrace{\begin{bmatrix} 0 & r & 0 & 0 \\ r & 0 & 0 & 0 \\ 0 & 0 & 0 & r \\ 0 & 0 & r & 0 \end{bmatrix}}_{\text{action } c}$$

Given the partition $\Delta = \{\{P_1, P_2\}, \{P_3, P_4\}\}$ on the state-space of P_1, we can construct the following aggregated rate matrices for each action, using (4.8):

$$\underbrace{\begin{bmatrix} 0 & 0.5 \times \top \\ 0 & 0 \end{bmatrix}}_{\text{action } a} \quad \underbrace{\begin{bmatrix} 0 & 0 \\ 0.5 \times \top & 0 \end{bmatrix}}_{\text{action } b} \quad \underbrace{\begin{bmatrix} r & 0 \\ 0 & r \end{bmatrix}}_{\text{action } c}$$

The matrices above cannot be used in the Kronecker representation directly however. The action types a and b are shared, so the corresponding rate matrices have to be broken down to apparent rate and probability matrices. The apparent rate matrix can be easily derived by constructing a diagonal matrix whose entries are the row sums of the corresponding rate matrix.

$$\underbrace{\underbrace{\begin{bmatrix}0 & 0.5\times\top \\ 0 & 0\end{bmatrix}}_{\text{rate matrix}} \xrightarrow{\text{row sums}} \underbrace{\begin{bmatrix}0.5\times\top & 0 \\ 0 & 0\end{bmatrix}}_{\text{apparent rate matrix}}}_{\text{action } a} \quad \underbrace{\underbrace{\begin{bmatrix}0 & 0 \\ 0.5\times\top & 0\end{bmatrix}}_{\text{rate matrix}} \xrightarrow{\text{row sums}} \underbrace{\begin{bmatrix}0 & 0 \\ 0 & 0.5\times\top\end{bmatrix}}_{\text{apparent rate matrix}}}_{\text{action } b}$$

Using the aggregated matrices above, we can produce the aggregated versions for the probability matrices for the shared actions a and b. The apparent rate matrices have to be inverted and subsequently multiplied by the corresponding transition rate matrices. Note that we have only kept the weights of the unspecified rates, otherwise the matrix inversion cannot be defined. For the same reason, we have added “ones” at the zero entries of the diagonal in the apparent rate matrices.

$$\underbrace{\underbrace{\begin{bmatrix}0.5 & 0 \\ 0 & 1\end{bmatrix}^{-1}}_{\text{apparent rate matrix}} \times \underbrace{\begin{bmatrix}0 & 0.5 \\ 0 & 0\end{bmatrix}}_{\text{rate matrix}} = \underbrace{\begin{bmatrix}0 & 0.25 \\ 0 & 0\end{bmatrix}}_{\text{probability matrix}} \xrightarrow{\text{normalise}} \underbrace{\begin{bmatrix}0 & 1 \\ 0 & 0\end{bmatrix}}_{\text{probability matrix}}}_{\text{action } a}$$

$$\underbrace{\underbrace{\begin{bmatrix}1 & 0 \\ 0 & 0.5\end{bmatrix}^{-1}}_{\text{apparent rate matrix}} \times \underbrace{\begin{bmatrix}0 & 0 \\ 0.5 & 0\end{bmatrix}}_{\text{rate matrix}} = \underbrace{\begin{bmatrix}0 & 0 \\ 0.25 & 0\end{bmatrix}}_{\text{probability matrix}} \xrightarrow{\text{normalise}} \underbrace{\begin{bmatrix}0 & 0 \\ 1 & 0\end{bmatrix}}_{\text{probability matrix}}}_{\text{action } b}$$

Step 2: Apply Kronecker Operators The aggregated version of component P_1, let that be P_{12}, is described by the following matrices:

$$\underbrace{\underbrace{\begin{bmatrix}0.5\times\top & 0 \\ 0 & 0\end{bmatrix}}_{\text{apparent rate matrix}} \quad \underbrace{\begin{bmatrix}0 & 1 \\ 0 & 0\end{bmatrix}}_{\text{probability matrix}}}_{\text{action } a} \quad \underbrace{\underbrace{\begin{bmatrix}0 & 0 \\ 0 & 0.5\times\top\end{bmatrix}}_{\text{apparent rate matrix}} \quad \underbrace{\begin{bmatrix}0 & 0 \\ 1 & 0\end{bmatrix}}_{\text{probability matrix}}}_{\text{action } b} \quad \underbrace{\begin{bmatrix}r & 0 \\ 0 & r\end{bmatrix}}_{\text{action } c}$$

For the Q_1 component we have no local activities, meaning that the individual rate matrix will be the null matrix. For the shared actions a and b, we have the following matrices:

$$\underbrace{\underbrace{\begin{bmatrix} r_a & 0 \\ 0 & 0 \end{bmatrix}}_{\text{apparent rate matrix}} \quad \underbrace{\begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}}_{\text{probability matrix}}}_{\text{action } a} \quad \underbrace{\underbrace{\begin{bmatrix} 0 & 0 \\ 0 & r_b \end{bmatrix}}_{\text{apparent rate matrix}} \quad \underbrace{\begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}}_{\text{probability matrix}}}_{\text{action } b}$$

We can then apply Equation (4.19) in order to produce the generator matrix of the composition $P_{12} \underset{\{a,b\}}{\bowtie} Q_1$. For convenience, we shall construct the transition rate matrix rather than the generator matrix, so as not to have to worry about the row sums being zero. By summing over all actions, we have the following transition rate matrix R:

$$R = \underbrace{\underbrace{\begin{bmatrix} r & 0 \\ 0 & r \end{bmatrix}}_{P_{12}\text{ component}} \oplus \underbrace{\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}}_{Q_1\text{ component}}}_{\text{individual rate matrices}}$$

$$+ \underbrace{\underbrace{\left(\underbrace{\begin{bmatrix} 0.5 \times \top & 0 \\ 0 & 0 \end{bmatrix}}_{P_{12}\text{ component}} \circledmin \underbrace{\begin{bmatrix} r_a & 0 \\ 0 & 0 \end{bmatrix}}_{Q_1\text{ component}} \right)}_{\text{apparent rate matrices}} \times \underbrace{\left(\underbrace{\begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}}_{P_{12}\text{ component}} \otimes \underbrace{\begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}}_{Q_1\text{ component}} \right)}_{\text{probability matrices}}}_{\text{action } a}$$

$$+ \underbrace{\underbrace{\left(\underbrace{\begin{bmatrix} 0 & 0 \\ 0 & 0.5 \times \top \end{bmatrix}}_{P_{12}\text{ component}} \circledmin \underbrace{\begin{bmatrix} 0 & 0 \\ 0 & r_b \end{bmatrix}}_{Q_1\text{ component}} \right)}_{\text{apparent rate matrices}} \times \underbrace{\left(\underbrace{\begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}}_{P_{12}\text{ component}} \otimes \underbrace{\begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}}_{Q_1\text{ component}} \right)}_{\text{probability matrices}}}_{\text{action } b}$$

For each action we have a matrix multiplication of the Kronecker minimum for the apparent rate matrices, and the Kronecker product of the corresponding action probability matrices. Note that the unspecified rates $\top$ are now specified because of the effect of the Kronecker minimum as in Definition 16, and the minimum operator in

Equation (4.11). If we calculate those, we obtain:

$$R = \underbrace{\begin{bmatrix} r & 0 & 0 & 0 \\ 0 & r & 0 & 0 \\ 0 & 0 & r & 0 \\ 0 & 0 & 0 & r \end{bmatrix}}_{\text{individual rate matrix}} + \underbrace{\underbrace{\begin{bmatrix} r_a & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}}_{\text{apparent rate matrix}} \times \underbrace{\begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}}_{\text{probability matrix}}}_{\text{action } a}$$

$$+ \underbrace{\underbrace{\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & r_b \end{bmatrix}}_{\text{apparent rate matrix}} \times \underbrace{\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}}_{\text{probability matrix}}}_{\text{action } b} = \begin{bmatrix} r & 0 & 0 & r_a \\ 0 & r & 0 & 0 \\ 0 & 0 & r & 0 \\ r_b & 0 & 0 & r \end{bmatrix}$$

The resulting matrix highlights the issue discussed in the previous section. If we examine R more carefully, we observe that it consists of three communicating classes: the first class involves the first and the fourth state, while the other two classes involve the second and the third state correspondingly. The inclusion of unreachable states has caused R to be a 4×4 matrix, whose size is equal to the original non-aggregated state-space in Figure 4.2. Nevertheless, the Kronecker representation can be useful if the reachable state-space approaches than the Cartesian product of the components involved.

4.4.2 A Structured Operational Semantics Characterisation

In this section we discuss an alternative approach to produce compositions of aggregated components. Our objective is to overcome the limitations of the Kronecker representation discussed in the previous section, and apply compositional aggregation to a wider family of models. Since the actual state-space for any cooperation is given by the operational semantics of the language outlined in Figure 2.1, it is a reasonable idea to investigate how component aggregation and cooperation of aggregated components can be defined in terms of the original PEPA semantics.

4.4.2.1 Activity Aggregation

We shall formally describe the aggregated component in terms of the initial one; this will allow us to manipulate the operational semantics of PEPA in order to produce compositions of such components. We know that the labelled transition system of some component P corresponds to a collection of rate matrices for the different actions of P. In Section 4.3, we have discussed how we can aggregate these matrices given some partition of the state-space. Next we shall look into the interpretation of these new states and new transitions that have been produced.

States that belong to the same class will collapse into a single state. If Δ_P is a partition on P, then state aggregation can be formally described by the following rule:

$$\frac{A \in \Delta_P \quad P \in A}{P_A \equiv P}$$

Informally, the rule says that if A is a class in Δ_P, then there is a state P_A in the aggregated model which is interpreted as being in any of the states in the class. The aggregate state has been conveniently labelled with the class, however the interpretation would be no different with a different label. The important thing is that we miss the initial fine-grained information about the individual states in A. It is therefore inevitable to assume that all of the included states are equally probable.

The next step is to investigate how the activities are affected in the aggregated state-space. First, we consider the case of transitioning from an aggregated to a single state:

$$\frac{P \xrightarrow{(\alpha,r)} P'}{P_A \xrightarrow{(\alpha,r/|A|)} P'} \; (P \in A)$$

Intuitively, the rule states that P_A is able to perform any of the activities of the components included in A. The rate however of each one of those activities is a fraction of their original rate. In fact, since we cannot discriminate between the states in A, all we know about P_A is that it is interpreted as P with probability $1/|A|$. Thus, whenever there is a rate r from $P \in A$ to P', the corresponding rate for P_A has to be adjusted accordingly.

Another rather simple case is when transitioning from a single state to an aggregated state. The rule that follows describes how an activity of a component P is distributed to a class $A' \in \Delta_P$.

$$\frac{P \xrightarrow{(\alpha,r)} P'}{P \xrightarrow{(\alpha,r)} P_{A'}} \; (P' \in A')$$

According to the rule above, P is still able to perform the same activity at the same rate. The information about the exact target state may have been lost as we only know the target class, but this does not affect the rate of the activity.

Finally, the two cases above considered individually can be summarised in the following rule:

$$\frac{P \xrightarrow{(\alpha,r)} P'}{P_A \xrightarrow{(\alpha,r/|A|)} P_{A'}} \; (P \in A, P' \in A') \tag{4.20}$$

If we apply the rule in (4.20) for every $P \in A$ and every $P' \in A'$, we essentially have multiple occurrences of the same activity, probably with different rates. This is totally acceptable as on the basis of the operational semantics PEPA models can be defined as multi-transition systems. It is reasonable however to collapse all of these activity instances into a single activity. Then the total rate from P_A to $P_{A'}$ for action type α will be:

$$q(P_A, P_{A'}, \alpha) = \frac{\sum_{P \in A} \sum_{P' \in A'} q(P, P', \alpha)}{|A|} \tag{4.21}$$

where $q(P, P', \alpha)$ is the rate of the activity of type α from P to P'. The rate expression above is equivalent to Equation (4.8) used in the matrix-based aggregation of PEPA components in Section 4.3.

Regarding passive activities, these will be handled as discussed in Section 4.3. More specifically, we assume that any unspecified rate is associated with a weight $w > 0$. Moreover, approximate aggregation will have effect on the weights, rather than on the corresponding unspecified rates. Finally, it is required that a certain action type is either globally active or globally passive for the derivative set of a given component.

4.4.2.2 The Aggregated Derivation Graph

So far, we have characterised component aggregation by a structured operational semantics approach. In this section, we investigate the effects that aggregation has on the PEPA semantics. We derive the aggregated semantics for each of the PEPA operators introduced in [48]. More specifically, we rewrite the operational semantics rules of Figure 2.1 in the style that we have defined activities for aggregated components in (4.20). Eventually, we shall see that the aggregated semantics proposed are equivalent to the combined effect of the rule in (4.20) with the original PEPA semantics.

Prefix According to the original specification of the prefix operator, whenever a component $(\alpha, r).P$ carries out an activity (α, r), it subsequently behaves as P. If now the target component is a class $A \in \Delta_P$, then the rate r should be distributed to the class:

$$\frac{(\alpha, r).P \xrightarrow{(\alpha,r)} P}{(\alpha, r).P_A \xrightarrow{(\alpha,r)} P_A} \; (P \in A) \tag{4.22}$$

The rule above describes the combined effect of Equation (4.20) and the prefix rule in the original PEPA semantics.

Choice The choice operator implies that a component $P+Q$ will behave either as P or Q. Let us first consider the case where there is an activity from P to P', assuming a partition Δ_P such that $A, A' \in \Delta_P$. Since the Q component is not selected, it does not matter whether it is aggregated or not. Regarding the P component, the activity in the aggregated state-space will be formed in a way similar to the rule in Equation (4.20):

$$\frac{P \xrightarrow{(\alpha,r)} P'}{P_A + Q \xrightarrow{(\alpha,r/|A|)} P_{A'}} \ (P \in A, P' \in A') \tag{4.23}$$

Similarly, whenever there is an activity from Q to Q', assuming a partition Δ_Q such that $B, B' \in \Delta_Q$, we have the following rule:

$$\frac{Q \xrightarrow{(\alpha,r)} Q'}{P + Q_B \xrightarrow{(\alpha,r/|B|)} Q_{B'}} \ (Q \in B, Q' \in B') \tag{4.24}$$

Unsynchronised Cooperation An unsynchronised cooperation is defined over an action type a which does not belong in the cooperation set L. Any component, either aggregated or not, may simply carry out any activity without affecting the state of other components. Therefore, starting from the rule in (4.20), we can consider the following rules for unsynchronised parallel composition:

$$\frac{P \xrightarrow{(\alpha,r)} P'}{P_A \underset{L}{\bowtie} Q \xrightarrow{(\alpha,r)} P_{A'} \underset{L}{\bowtie} Q} \ (\alpha \notin L, P \in A, P' \in A') \tag{4.25}$$

$$\frac{Q \xrightarrow{(\alpha,r)} Q'}{P \underset{L}{\bowtie} Q_B \xrightarrow{(\alpha,r)} P \underset{L}{\bowtie} Q_{B'}} \ (\alpha \notin L, Q \in B, Q' \in B') \tag{4.26}$$

Synchronised Cooperation Starting from the synchronised cooperation rule of Figure 2.1, we shall modify it so as to admit aggregated components. Let Δ_P be a partition on P, where $A, A' \in \Delta_P$, and similarly Δ_Q be a partition on the state-space of Q, where $B, B' \in \Delta_Q$. We also assume that $P \in A$ and $P' \in A'$, while in the same way $Q \in B$ and $Q' \in B'$. Thus whenever there is a synchronised activity from P to P' and from Q to Q', for the corresponding aggregated states the activity will be defined as follows:

$$\frac{P \xrightarrow{(\alpha,r_1)} P' \quad Q \xrightarrow{(\alpha,r_2)} Q'}{P_A \underset{L}{\bowtie} Q_B \xrightarrow{(\alpha,R)} P_{A'} \underset{L}{\bowtie} Q_{B'}} \quad (\alpha \in L, P \in A, P' \in A', Q \in B, Q' \in B') \tag{4.27}$$

where

$$R = \frac{r_1}{|A| \times r_\alpha(P_A)} \frac{r_2}{|B| \times r_\alpha(Q_B)} \min(r_\alpha(P_A), r_\alpha(Q_B)) \tag{4.28}$$

According to (4.20), we know that the rate of this particular activity for P_A will be $r_1/|A|$, while for Q_B it will be $r_2/|B|$. The rate expression above is identical to the original in Figure 2.1, aside from the fact that we have modified the rates according to the aggregation rule in (4.20). The apparent rates can be easily derived; given an aggregated component P_A for some action type α, the apparent rate will be:

$$r_\alpha(P_A) = \sum_{A' \in \Delta_P} q(P_A, P_{A'}, \alpha) \tag{4.29}$$

We have already stated that an action type will be consistently active or passive for the entire derivative set of any component. The resulting apparent rate will be either a positive real number or unspecified, just as happens for non-aggregated components.

One last thing to consider is that the rule in (4.27) will be applied for every possible combination of states P, P', Q and Q' that belong to the classes A, A', B and B' correspondingly. The total rate of transitioning from $P_A \underset{L}{\bowtie} Q_B$ to $P_{A'} \underset{L}{\bowtie} Q_{B'}$ will be:

$$q(P_A \underset{L}{\bowtie} Q_B, P_{A'} \underset{L}{\bowtie} Q_{B'}, \alpha) = \frac{\sum_{P \in A} \sum_{P' \in A'} q(P, P', \alpha)}{|A| \times r_\alpha(P_A)} \times \frac{\sum_{Q \in B} \sum_{Q' \in B'} q(Q, Q', \alpha)}{|B| \times r_\alpha(Q_B)} \times \min(r_\alpha(P_A), r_\alpha(Q_B))$$

Regarding the equation above, we observe that the first two terms aside from the apparent rates are equal to the collapsed activity rates for the aggregated components P_A and Q_A, as given by Equation (4.21). So the total rate of a synchronised transition can be further simplified as follows:

$$q(P_A \underset{L}{\bowtie} Q_B, P_{A'} \underset{L}{\bowtie} Q_{B'}, \alpha) = \frac{q(P_A, P_{A'}, \alpha)}{r_\alpha(P_A)} \times \frac{q(Q_B, Q_{B'}, \alpha)}{r_\alpha(Q_B)} \times \min(r_\alpha(P_A), r_\alpha(Q_B)) \tag{4.30}$$

The rate expression for aggregated components in (4.30) is essentially identical to the expression used for non-aggregated synchronised activities. Therefore, activity aggregation as defined in Section 4.4.2.1 allows us to employ the operational semantics in Figure 2.1 to construct the derivation graph for any composition of aggregated components.

Hiding Recall that a component P/L behaves as P, except that any activity of any action type in L will be hidden, meaning that it will be considered as a local activity. Combining the semantics of rules for the hiding operator in Figure 2.1 with the rule in Equation (4.20), we obtain the following operators that describe the combined effect:

$$\frac{P \xrightarrow{(\alpha,r)} P'}{P_A/L \xrightarrow{(\alpha,r/|A|)} P_{A'}/L} \; (\alpha \notin L, P \in A, P' \in A') \tag{4.31}$$

$$\frac{P \xrightarrow{(\alpha,r)} P'}{P_A/L \xrightarrow{(\tau,r/|A|)} P_{A'}/L} \; (\alpha \in L, P \in A, P' \in A') \tag{4.32}$$

4.4.2.3 A Worked Example

We shall now revisit the example of Section 4.4.1.4, in order to demonstrate how the operational semantics approach responds to models whose reachable state-space is smaller than the Cartesian product. Recall that the P_1 has to be aggregated with respect to a partition $\Delta = \{\{P_1,P_2\},\{P_3,P_4\}\}$. Applying the standard PEPA semantics, for the P_1 component only, will result in the graph of Figure 4.3.

$$\begin{tikzcd}
P_1 \arrow[rr, "(a,\top)"] \arrow[d, "(c,r)"', shift right] & & P_3 \arrow[d, "(c,r)"', shift right] \\
P_2 \arrow[u, bend right, "(c,r)"'] & & P_4 \arrow[u, bend right, "(c,r)"'] \arrow[ll, "(b,\top)"']
\end{tikzcd}$$

Figure 4.3: Derivation graph for P_1

It is preferable to aggregate the state-space of P_1 first, and then apply the rules for the composition with the Q_1 component. In this way, we avoid unnecessarily large intermediate components. According to Δ, the states P_1 and P_2 are collapsed into a single state; let that be P_{12}. Similarly, P_3 and P_4 are collapsed into P_{34}.

The next step will be to apply the rule in (4.20) for states P_1 and P_2, and their amalgamation P_{12}. This will add three activities in the aggregated derivation graph:

$(c,r/2).P_{12}$, $(c,r/2).P_{12}$ and $(a,\top/2).P_{34}$. Similarly, applying rule (4.20) on P_3 and P_4 will add the activities: $(c,r/2).P_{34}$, $(c,r/2).P_{34}$ and $(b,\top/2).P_{12}$. The aggregated derivation graph for the component is pictured in Figure 4.4. Activities of the same type having the same source and target states can be collapsed into a single activity whose rate is the sum of the corresponding activity rates.

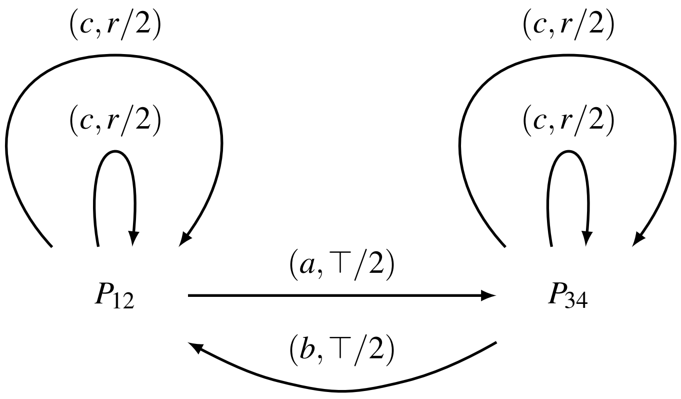


Figure 4.4: Derivation graph for P_{12}

Finally, using the standard operational semantics of PEPA, we can create the state-space for the composition $P_1 \underset{\{a,b\}}{\bowtie} Q_1$ as depicted in Figure 4.5. It is easy to see that the same graph would have been produced if we first derive the full graph for $P_1 \underset{\{a,b\}}{\bowtie} Q_1$, and subsequently apply the aggregated operational semantics of Section 4.4.2.2.

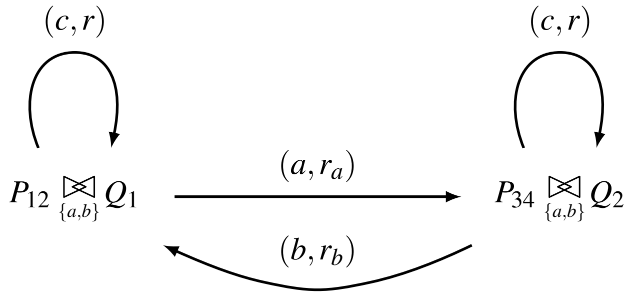

(c,r)
(c,r)
$P_{12} \underset{\{a,b\}}{\bowtie} Q_1$
(a,r_a)
$P_{34} \underset{\{a,b\}}{\bowtie} Q_2$
(b,r_b)

Figure 4.5: Derivation graph for $P_{12} \underset{\{a,b\}}{\bowtie} Q_1$

4.5 Experimentation on Multi-Scale Systems

In this section, we demonstrate the potential of the compositional approximate aggregation through an example. We shall consider models featuring high-population components, as even simple model descriptions can lead to very large state-spaces. In particular, multi-scale models are of interest since more efficient approaches such as fluid flow approximation [49] are not as readily applicable, because they make an assumption of continuity which is strained at low population numbers. So we consider a peer-to-peer system that involves large numbers of peers that communicate with each other with the help of an indexing server, as described in the following PEPA model:

$$\begin{aligned}
PeerA &\overset{def}{=} (localAction_A, r_{localA}).PeerA_{local} \\
&\quad + (lookup_B, \top).PeerA_{lookup} \\
PeerA_{local} &\overset{def}{=} (finish_A, r_{finishA}).PeerA \\
PeerA_{lookup} &\overset{def}{=} (cache_A, r_{cacheA}).PeerA_{local} \\
&\quad + (exchange, r_{exchangeA}).PeerA \\
PeerB &\overset{def}{=} (localAction_B, r_{localB}).PeerB_{local} \\
&\quad + (lookup_A, \top).PeerB_{lookup} \\
PeerB_{local} &\overset{def}{=} (finish_B, r_{finishB}).PeerB \\
PeerB_{lookup} &\overset{def}{=} (cache_B, r_{cacheB}).PeerB_{local} \\
&\quad + (exchange, r_{exchangeB}).PeerB
\end{aligned}$$

Our system involves two classes of peers which exchange data pairwise. Both types of peers have some local functionality and a shared activity called *exchange*. Moreover, a peer will have to look up other peers in an indexing server before proceeding to any data exchange.

$$\begin{aligned}
Index &\overset{def}{=} (lookup_A, r_{lookupA}).Index_{busyA} \\
&\quad + (lookup_B, r_{lookupB}).Index_{busyB} \\
&\quad + (fail, r_{fail}).Index_{broken} \\
Index_{busyA} &\overset{def}{=} (refresh, r_{refresh}).Index \\
&\quad + (fail, r_{fail}).Index_{broken} \\
Index_{busyB} &\overset{def}{=} (refresh, r_{refresh}).Index \\
&\quad + (fail, r_{fail}).Index_{broken} \\
Index_{broken} &\overset{def}{=} (repair, r_{repair}).Index
\end{aligned}$$

In Chapter 3, we have defined two different approaches for approximate Markov chain aggregation, which we shall now compare in a compositional setting. The quasi-lumpability approach described in Sect. 3.2 involves applying a clustering algorithm

Table 4.1: The rate values used in the examples

Variable Name	Value	Variable Name	Value	Variable Name	Value
r_{localA}	5	r_{localB}	2	$r_{lookupA}$	10
$r_{finishA}$	4	$r_{finishB}$	3	$r_{lookupB}$	10
r_{cacheA}	1	r_{cacheB}	2	r_{fail}	0.02
$r_{exchangeA}$	1	$r_{exchangeB}$	0.5	$r_{refresh}$	10
				r_{repair}	0.5

on the row entries of the transition probability matrix of a Markov chain. The NCD based approach discussed in Sect. 3.1 partitions the Markov chain according to the eigenvectors that correspond to the top eigenvalues of the probability matrix. For each one of the examples that follow, we explicitly note which components have been approximated and what compression ratio has been used. By the term "compression ratio", we refer to the ratio of the size of an aggregated component to its original size.

Once a nearly optimal partition of the state-space of a particular component is obtained using either of the two methods, that component is then approximately aggregated as described in Sect. 4.3. The final aggregated CTMC is produced after combining all of the components involved in the model. We have presented two distinct methods to combine aggregated components: a Kronecker representation approach in Section 4.4.1, and a structured operational semantics approach in Section 4.4.2. Both representations have been used to produce the experimental results that follow.

Eventually, we compare the transient and the steady-state behaviour of the initial model with those of the approximately aggregated models. The PRISM model checker [58], its sparse engine in particular, has been used for that purpose. The Gauss-Seidel method has been applied for computing the steady-state distribution, and the uniformisation method for the transient probabilities. The experiments have been performed in an Intel® Xeon$^{\text{TM}}$ E5410 @ 2.33GHz PC running Scientific Linux 6.

4.5.1 Compositional vs Global Aggregation

In this experiment we define a system small enough to compare the compositional approximate aggregation with a globally applied approach. The first system's structure is summarised in the following system equation, with cooperation sets $\mathcal{L} = \{exchange\}$ and $\mathcal{K} = \{lookup_A, lookup_B\}$.

$$System_{5:5:1} \overset{def}{=} PeerA[5] \underset{\mathcal{L}}{\bowtie} PeerB[5] \underset{\mathcal{K}}{\bowtie} Index$$

Table 4.2: Running times for $System_{5:5:1}$

	Original	Quasi-Lumpability (Compositional)	NCD (Compositional)	Quasi-Lumpability (Global)	NCD (Global)
Approximation	-	0.002 sec	0.002 sec	40 sec	40 sec
PRISM Loading	1.2 sec	0.25 sec	0.25 sec	0.4 sec	0.5 sec
Transient Solution[a]	1.7 sec	0.4 sec	0.4 sec	0.8 sec	0.8 sec
Steady-State Solution	0.1 sec	0.025 sec	0.025 sec	0.05 sec	0.05 sec
Total Time	3.0 sec	0.677 sec	0.677 sec	41.35 sec	41.35 sec
Number of States	1764	400	400	400	400

[a] 100 points: $0 \leq t \leq 2$

If we apply exact aggregation as described in [42], the number of states for the *PeerA*[5] and *PeerB*[5] components will be 21 (these would be 243 for each with no aggregation). Therefore, we distinguish the following cases:

i. *PeerA*[5] and *PeerB*[5] components are further reduced independently. The compression ratio used is 0.5 for both, resulting in a reduced chain of 400 states.

ii. Approximate aggregation is applied on the entire system's generator matrix. The compression ratio used was such that it results in a reduced chain of 400 states.

The quasi-lumpability and the NCD-based approaches have been applied in both a compositional and a global setting. Thus, we essentially have four approximate aggregation methods to compare, which are applied over two representations, based on either Kronecker algebra or structured operational semantics. For the model considered, the state-space reduction has been the same for the two representations. Table 4.2 summarises the running times for aggregating and solving the model. The solution time refers to the amount of time required by PRISM; this is broken down to the time needed to load the model, calculate the transient probabilities for 100 time-points, and calculate the steady-state probabilities. As expected, compositional aggregation requires a very small initial cost to reduce the model, in contrast to the global case.

As in Section 3.4.1, K-L divergence has been used to measure approximation quality. We shall first comment on the results of the Kronecker representation. Figure 4.6(a) summarises the K-L divergences at different times t, for the four approximate aggregation methods. Judging by the K-L divergences, global aggregation does not appear to be far superior to the compositional approaches. Although we do not claim that this statement generalises to every possible model, it seems reasonable to use compositional aggregation in order to produce a reasonable approximation of the original stochastic process. This argument is supported by the running times in Table 4.2.

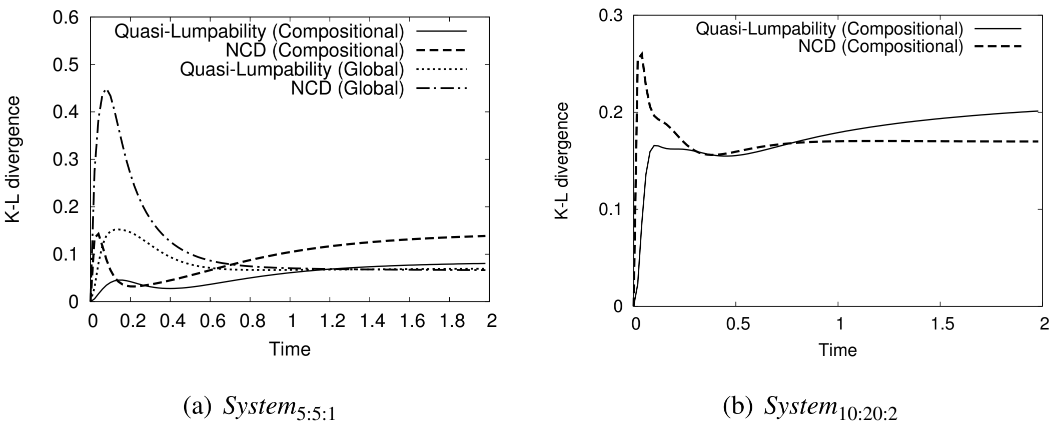


Figure 4.6: K-L divergences between the true and the approximate state-space distribution at different times, for various aggregation methods (Kronecker representation)

It is also interesting to compare how the quasi-lumpability and the NCD-based approaches behave. For $System_{5:5:1}$ in 4.6(a), both the global and the compositional setting appear to favour the quasi-lumpability approach. It is not safe to generalise this result for arbitrary models however. In fact, the results are rather contradictory for Figure 4.6(b), which depicts the K-L divergences for $System_{10:20:2}$ of the next section. For this larger model, the compositionally applied quasi-lumpability approach performs better than the NCD approach in the early stages of the stochastic process, but it becomes less accurate as steady-state behaviour is approached. In the general case, we cannot say that either the quasi-lumpability or the NCD-based approach provides significantly better approximation quality.

Regarding the structured operational semantics representation, the K-L divergence results are outlined in Figure 4.7, which are almost identical to the results for the Kronecker representation in Figure 4.6. As a matter of fact, both representations are just different interpretations of the same mechanism which involves PEPA semantics and approximate component aggregation. The structured operational semantics approach is more appropriate in a general context however, as no unreachable states are included in the generator matrix of the resulting CTMC.

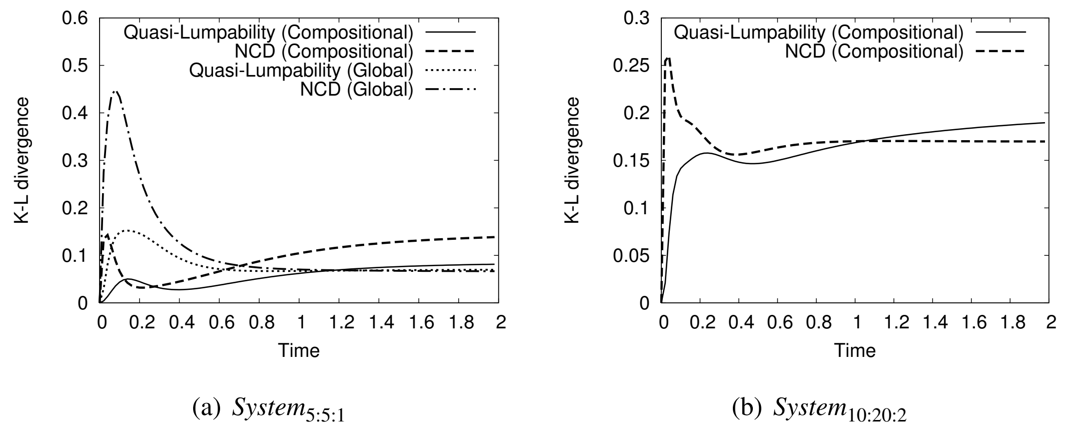


Figure 4.7: K-L divergences between the true and the approximate state-space distribution at different times, for various aggregation methods (Structured operational semantics representation)

4.5.2 Component Behaviour

This second example provides a more detailed view of component behaviour. The following system equation is considered, with cooperation sets $\mathcal{L} = \{exchange\}$ and $\mathcal{K} = \{lookup_A, lookup_B\}$.

$$System_{10:20:2} \stackrel{def}{=} PeerA[10] \underset{\mathcal{L}}{\bowtie} PeerB[20] \underset{\mathcal{K}}{\bowtie} Index[2]$$

We report the results for the structured operational semantics representation only. The Kronecker representation results are practically identical, as can be seen by comparing the K-L divergences in Figures 4.6(b) and 4.7(b).

If we apply exact aggregation as described in [42], the $PeerA[10]$ component will have 66 states, while $PeerB[20]$ will have 231 states (these would be 59,049 and 3,486,784,401 states with no aggregation). Although neither of the components is particularly large, their combination produces a large state-space. However, it is relatively easy to further reduce $PeerA[10]$ and $PeerB[20]$ independently. The compression ratio used is 0.5 for both components.

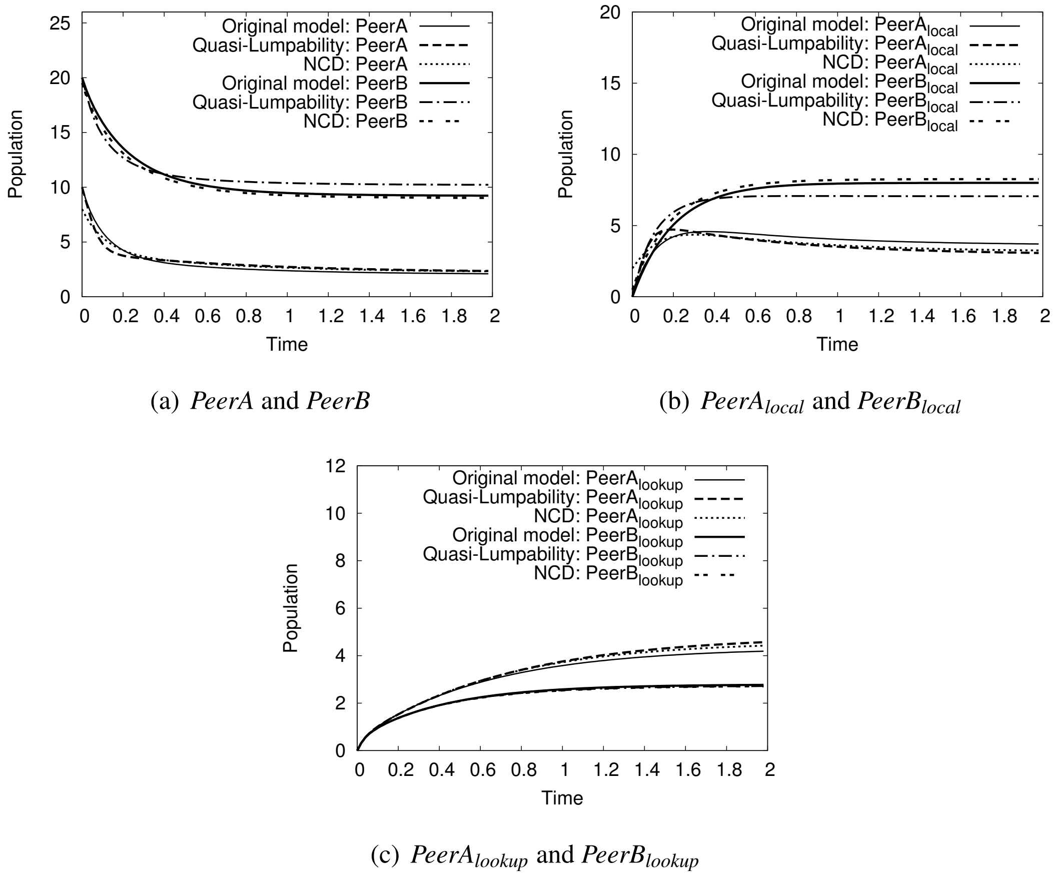
This approximation of individual components results in significant reduction of the total state-space. As can be seen in Table 4.3, this reduction required only a small initial cost, while it resulted in a considerable decrease of the analysis time. A global reduction of the state-space would be practically infeasible for a model of such size. Figure 4.8 depicts the evolution of the average populations of the model components that have been reduced. Those figures seem to be reasonable approximations of the

Table 4.3: Running times for $System_{10:20:2}$

	Original	Quasi-Lumpability (Compositional)	NCD (Compositional)
Approximation	-	1 sec	1 sec
PRISM Loading	320 sec	70 sec	70 sec
Transient Solution[b]	300 sec	75 sec	75 sec
Steady-State Solution	20 sec	4 sec	4 sec
Total Time	640 sec	150 sec	150 sec
Number of States	152460	37950	37950

[b] 100 points: $0 \leq t \leq 2$

original model's average behaviour. It would also be interesting though to look at the behaviour of the components that have not been approximated. Figure 4.9 depicts the evolution of the average *Index* populations. Both quasi-lumpability and NCD-based approach result in approximations very close to the original solution.

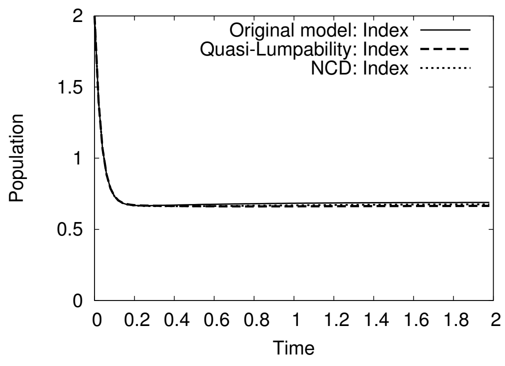


(a) *PeerA* and *PeerB*

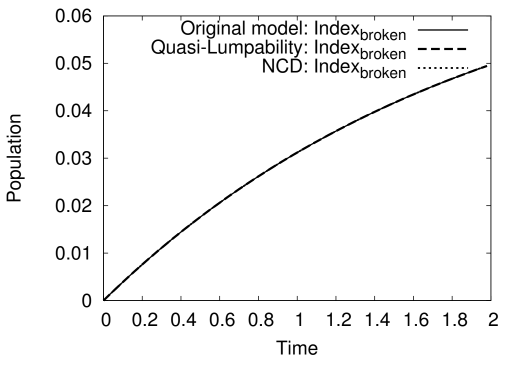
(b) $PeerA_{local}$ and $PeerB_{local}$

(c) $PeerA_{lookup}$ and $PeerB_{lookup}$

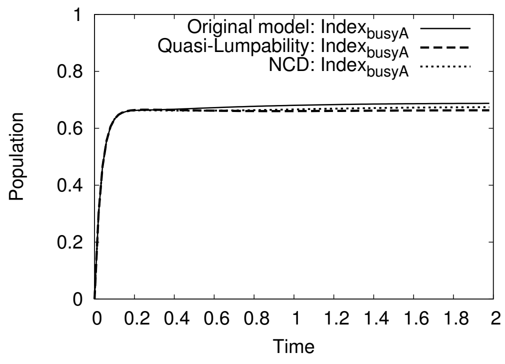
Figure 4.8: Evolution of the average *PeerA* and *PeerB* populations for $System_{10:20:2}$



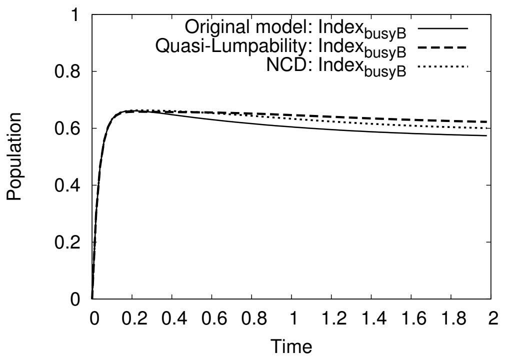
(a) *Index*



(b) *Index$_{broken}$*



(c) *Index$_{busyA}$*



(d) *Index$_{busyB}$*

Figure 4.9: Evolution of the average *Index* populations for $System_{10:20:2}$

4.5.3 Experimentation with the System Size

By imposing a certain compression ratio, we essentially decide the size of the aggregated component, and subsequently the size of the entire aggregated model. In this experiment, we explore how approximation quality is affected by different compression ratios for the models considered in this chapter. Moreover, we examine whether the effect of a particular compression ratio is different when aggregation is applied to systems of different size.

We consider the systems $System_{5:5:1}$ and $System_{10:20:2}$, which have been used in the previous sections. This time, we look into the relative errors regarding the average component populations at steady-state. The relative error is calculated as follows:

$$\eta = \frac{|v - v_{approx}|}{v} \tag{4.33}$$

where v is the true value for the component population, and v_{approx} is the corresponding approximate value. The values that we report are the averages of the relative errors for all the PEPA components involved in $System_{5:5:1}$ and $System_{10:20:2}$.

Table 4.4(a) outlines the experimentation results for $System_{5:5:1}$. In the first column, we can see the compression ratios used for component aggregation. The second column shows the total state-space size of the aggregated Markov chain. The components that have been aggregated are $PeerA[5]$ and $PeerB[5]$ with ratio that varies between 0.3 and 0.7. For each aggregation case, we report the average relative errors for the quasi-lumpability and the NCD-based approach. In the general case, one would expect that the more the size of the model is reduced, the less accurate the results will be. For the model considered, we can see that we have higher error values as we decrease the size of the aggregated model, regardless of the aggregation approach used in each case.

Table 4.4: Experimentation with different compression ratios

(a) $System_{5:5:1}$

Component Compression Ratio[c]	Number of States[d]	Average Relative Error at Steady-State: Quasi-Lumpability	Average Relative Error at Steady-State: NCD
0.7	784	0.115	0.055
0.5	400	0.125	0.099
0.3	144	0.151	0.144

[c] Aggregation has been applied to $PeerA[5]$ and $PeerB[5]$ independently

[d] The original model has 1764 states

(b) $System_{10:20:2}$

Component Compression Ratio[e]	Number of States[f]	Average Relative Error at Steady-State: Quasi-Lumpability	Average Relative Error at Steady-State: NCD
0.7	74060	0.057	0.021
0.5	37950	0.078	0.044
0.3	13110	0.108	0.094

[e] Aggregation has been applied to $PeerA[10]$ and $PeerB[20]$ independently

[f] The original model has 152460 states

The same set of compression ratios is used for the $System_{10:20:2}$, whose results are outlined in Table 4.4(b). Again, we observe that the errors tend to be larger as the size of the approximate model is decreased. Another interesting observation is that the relative errors in Table 4.4(b) are significantly smaller than the corresponding values for the smaller system in Table 4.4(a). Although we have experimented with the same compression ratios for both models, approximation quality has been better for the larger model. Apparently, a larger model is more likely to be amenable to approximate state-space aggregation.

As a final remark, it appears there is a trade-off between approximation quality and state-space reduction. Nevertheless, this result should not be generalised for arbitrary models. Sometimes it should be possible to achieve a state-space reduction that is more accurate than other configurations that involve more states. A characteristic example is the case where the model in question is lumpable; there would be no approximation error, as the aggregation would be exact.

4.6 Summary

Although approximate Markov chain aggregation is not a new concept, it has not been particularly popular in the field of Markovian modelling, since an explicit representation of the transition matrix is typically required. Using approximate Markov chain aggregation as discussed in Chapter 3, we were able to reduce the local state-space of the labelled CTMCs that correspond to PEPA components.

Component aggregation has to rely on some notion of equivalence between components. We have developed a modified version of strong equivalence as appeared in [48]. Our version has been defined over the set of transition matrices that characterise a component's behaviour as a labelled transition system. Given a partition on a component state-space, the individual rate matrix as well as the rate matrices for all shared actions have to be lumpable. We have suggested that the partitioning approaches of Chapter 3 can be applied to the individual rate matrix, in order to obtain a partition on the component state-space. Regarding the applicability of such an approach, it is noted that the approximated components are required to have a set of shared actions that is relatively small when compared to their set of individual actions. That would mean that the individual rate matrix is dense enough to apply a partitioning algorithm on it. Therefore, our approach is mostly applicable to models that can be decomposed to weakly dependent components.

In order to combine aggregated PEPA components in a way compatible to the original PEPA semantics, we have resorted to two alternatives. First, we have explored in more detail the Kronecker representation that we have proposed in [72], and we have highlighted a limitation, according to which, unreachable states may be involved in the generator matrix. An new approach to produce compositions of aggregated components has been also proposed, which involves the definition of appropriate structured operational semantics to produce an aggregated derivation graph.

Our multi-scale example demonstrated the potential of compositional approximate aggregation, as it produced reasonable approximations of the models considered. Most importantly, compositional aggregation resulted in a great reduction of the state-space size with a small initial cost for aggregating the PEPA components, in contrast with aggregating the entire Markov chain.

Chapter 5

Stochastic Simulation via Trajectory Sampling

The use of Markovian process algebras in biochemical modelling provides a formal context for representing biological systems with stochastic behaviour. A formal specification often provides a deep insight into critical aspects of the modelling process. For example, Clark et al. [25] applied a static analysis on Bio-PEPA components that helped identifying flaws in the model description, based on the notion of conservation of mass.

Nevertheless, in a biochemical context it is very difficult to apply many of the analysis methods that are available for a process algebra such as PEPA. Remember that PEPA models are collections of components with finite state space. On the other hand, a Bio-PEPA component represents population counts for a chemical species. Molecular populations are often unbounded, thus Bio-PEPA components do not necessarily take values in a finite set. This often results in Markov chains whose state-space is unbounded or just too large to produce an explicit representation. Even a compositional approach to construct the state-space, in the style of Kronecker representation for PEPA, would not be particularly useful, as Bio-PEPA components tend to be highly coupled. The transitions are dictated by reactions, whose rates are determined by the global state of the system rather than the local state of the component.

Stochastic simulation is still a relevant approach to explore the properties of chemical reaction systems described by Bio-PEPA models, as no explicit representation of the entire Markov chain is required. An introduction to Markov chain simulation can be found in Section 2.4. In short, simulating a stochastic process produces trajectories, which are realisations of the stochastic process, a concept similar to sampling from random variables. Generating a large number of such realisations allows us to produce estimates for the state probabilities at different times. In particular, a Markov chain trajectory is a sequence of states and transitions that realises a random walk over the state-space.

In the lifetime of a typical biochemical system, a huge number of reactions need to be simulated. Therefore, the generation of a single trajectory typically involves generating a large number of stochastic events, a fact that motivated research towards accelerated stochastic simulation approaches that are either exact or approximate. Exact approaches simulate every single reaction occurring in a system, but also make use of appropriate data structures in order to generate the simulation events efficiently. An approximate simulation method skips some of the simulation events, resulting in a significantly faster process when compared to exact methods. A review on simulation approaches can be found in Section 2.4.

In this chapter, we present an accelerated simulation algorithm that can be characterised as almost exact, in the sense that it produces all of the transitions involved in the realisation of a stochastic process. We call our method *Trajectory Sampling Simulation* (TSS) as it samples from the distribution of state sequences and the distribution of time given some particular sequence. Sampling from the trajectory space rather than the transition space means that we need to generate fewer random numbers, which is an operation that is typically computationally expensive. Sampling from the time distribution involves approximating the exponential distributions that govern the sojourn times with a geometric distribution. A proper selection for the approximation parameters can ensure that the stochastic process simulated is almost identical to the simulation of the original Markov chain. Our approach does not rely on certain properties of the model and it can be used as an alternative to more efficient approaches when those are not applicable.

In Section 5.1, we focus on simulation approaches in the literature that are most closely related to our work. Section 5.2 discusses theoretical details and implementation issues regarding the TSS algorithm. Our simulation algorithms is experimentally evaluated in Section 5.3.

5.1 Related Work

Our simulation algorithm is a modification of the *direct method* (DM) [40]. The DM has been originally described in terms of systems of chemical reactions, which perfectly fit the context of a typical Bio-PEPA model. We present a generic description of the DM in Algorithm 3, in order to cast light on some important aspects of CTMC simulation.

Algorithm 3 The Direct Method

```
1: Initialise system state
2: while $t < t_{final}$ do
3:     Given that $M$ transitions are currently possible, calculate the transition rates $\lambda_m$, $\forall m \in \{1, 2, \ldots, M\}$
4:     Calculate $\lambda = \sum_{m=1}^{M} \lambda_m$, which is the rate of leaving the current state
5:     Draw sample $\tau$ from $Exp(\lambda)$
6:     Select next transition with probability: $p_m = \frac{\lambda_m}{\lambda}$
7:     Update time: $t \leftarrow t + \tau$
8:     Update state with effect of transition $m$
9: end while
```

The main source of inefficiency for the DM is the fact that the steps above have to be repeated many times. In terms of chemical reaction systems, each iteration of Algorithm 3 corresponds to a singe reaction that involves the molecules prescribed by the system specification. Typically, millions of reactions have to be simulated in order to generate a single trajectory. Moreover, a large number of trajectories is needed to have a sufficiently accurate approximation of the system properties.

Approximate simulation methods skip some of the iterations, and therefore do not monitor every single change in the system. This can be appropriate for many biological systems, where the state stands for molecular populations of the different chemical species. If the populations are high enough, then small variations will not particularly affect the reaction rates. It is therefore assumed that the reaction rates remain constant for a certain amount of time, and the state is updated according to the combined result of a number of reactions. For example, τ-*leaping* [41] advances time by a time interval τ, while the number of firings for each reaction is determined by a Poisson random variable. A similar approach is *K-leap* [10], however, the number of reaction firings is fixed to k. Instead, the time interval τ is determined by a Gamma distributed random variable which represents the total duration of k exponential events at the same rate. Such practices result in a significantly accelerated simulation process, however, they are only accurate if the reaction rates remain practically constant during τ. In fact, the applicability of such approaches depends on the properties of the model. Systems that involve low population components are liable to give erroneous results.

In this work, we are interested in developing an accelerated simulation algorithm whose performance and accuracy are independent of the model in question. If we want to position our approach against the literature, it can be characterised as exact, as no transitions are skipped. The applicability of exact methods does not rely on the model properties, because simulation is typically accelerated by appropriate data structures. For example, the optimised DM (ODM) [18] makes use of a dependency graph, in order to keep track of the reaction rates that have been actually affected by the last change of state, and thus unnecessary recalculations are avoided. In our approach, the source of optimisation is the reduction of the total number of random numbers generated. This practice does not prohibit the use of other exact simulation approaches in combination with trajectory sampling. As a matter of fact, our implementation is based on the ODM, which is used as a baseline for efficiency comparisons.

The concept of minimising random number generation has also appeared in the *K-skip method I* in [9], or simply K-skip. While their strategy for sampling from the state sequence distribution is similar, their approach for sampling from the time distribution is different. In order to reduce the random samples that determine the sojourn times, they approximate the sum of k exponential random variables with a Gamma distribution, assuming that the exit rates are similar for subsequent states, in a similar way to the K-leap method which is approximate. This assumption is reasonable for many bio-chemical systems, however it may introduce errors for some models as we demonstrate in the experiments section, while our approach can be generalised for arbitrary models. We have implemented K-skip following its description in the original paper, in order to produce some comparative results. We also highlight some computational issues not considered in [9] that arise from the fact that one random number is used to produce an entire trajectory.

5.2 Trajectory Sampling Simulation

As a simulation approach, trajectory sampling can be characterised as almost exact, in the sense that it can be arbitrarily precise; accuracy is controlled by a user-defined parameter. In the case of the DM, each step requires sampling from two distributions: the state distribution and the time distribution, both conditioned on the current state. In a similar way, TSS involves sampling from the distribution of state sequences. This reduces the number of random samples generated, a fact that implies a faster simulation algorithm. The algorithm is still exact, since no transitions are skipped. The same approach is extended to sample from the time distribution given some particular sequence. That is achieved by approximating the exponentially distributed sojourn times with a discrete random variable. Time discretisation allows us to consider the time distribution as a discrete state Markov chain, and therefore employ the TSS technique. This modification essentially renders our approach approximate, as part of the stochastic behaviour of the CTMC is suppressed. However, we have shown that our method in the limit converges to the solution of the original process, a fact that explains our use of the term "almost exact". We show that an appropriate selection of the approximation parameters can result in a behaviour very close to the original CTMC, and in a reasonable speed-up at the same time.

5.2.1 Random Variables: Transition vs Trajectory Point of View

We have seen that a CTMC (Definition 2) can be represented as a triple $(S, Q, \boldsymbol{\pi}_0)$, where S is a set of states. In order to facilitate discussion, we consider S to be finite, although it does not have to be so in the general case. Then $Q \in \mathbb{R}^{|S| \times |S|}$ will be a finite generator matrix, and $\boldsymbol{\pi}_0 \in \mathbb{R}^{|S|}$ is the initial probability vector.

A transition in a CTMC is associated with two random variables that depend on the current state $s \in S$; those are X_s that determines the next state, and L_s that determines the amount of time spent in s. The DM involves sampling from X_s and L_s to generate the next event. The X_s random variable follows a categorical distribution conditional on s, and its probability mass function is given by row s of the jump matrix P. We assume an ordering of states such as $s < s'$, if s corresponds to a row of the transition matrix with a smaller index than s'. If s_{k-1} is the state of the system after $k-1$ transitions, sampling from $X_{s_{k-1}}$ involves using a uniform sample $U \sim \mathcal{U}(0,1)$ and selecting the next state s_k with probability:

$$Pr(X_{s_{k-1}} = s_k) = Pr(a_{s_k} < U \leq b_{s_k}) \tag{5.1}$$

where b_{s_k} is the cumulative probability of state s_k given s_{k-1}, while a_{s_k} is the cumulative probability of the state that precedes s_k in the ordering:

$$a_{s_k} = \sum_{s_k{}' < s_k} P_{s_{k-1} s_k{}'} \qquad \text{and} \qquad b_{s_k} = \sum_{s_k{}' \leq s_k} P_{s_{k-1} s_k{}'} \tag{5.2}$$

In order to sample from $L_{s_{k-1}}$, we have to draw a new uniform sample $U \sim \mathcal{U}(0,1)$ and calculate the time $t_{s_{k-1}}$ spent in s_{k-1} as follows:

$$t_{s_{k-1}} = -\frac{\ln(1-U)}{\lambda_{s_{k-1}}} \tag{5.3}$$

From a trajectory point of view, the random variables are different. A CTMC trajectory involves a sequence of states and a sequence of positive numbers that represent the amount of time spent in each state. Let $\mathcal{S}_k$ be a collection that stands for the family of state sequences of length k. Therefore, we define $X_{\mathcal{S}_k}$ as the variable that represents the k-length sequence distribution. Given some particular sequence of states, namely $s_{0:k}$, its duration is represented by the $L_{s_{0:k}}$ random variable. Ideally, we would like to directly sample from $X_{\mathcal{S}_k}$ and $L_{s_{0:k}}$ to determine the state history and the time of the system after k transitions. Exact stochastic simulation algorithms actually sample from those distributions implicitly by advancing by one state at each event. In the sections that follow, we discuss how we can directly sample from the trajectory-related distributions, $X_{\mathcal{S}_k}$ and $L_{s_{0:k}}$.

5.2.2 Sampling from the State Sequence Distribution

The sampling from the state sequence distribution discussed in this section can be applied to both discrete and continuous time processes. Without loss of generality, we can assume that there is one initial state in some Markov chain. This will be the root of a tree whose paths represent all the possible state sequences. Each path of a tree with k levels corresponds to a sequence of $k+1$ states or k transitions. Then, the probability of a path can be defined as the product of the transitions involved:

$$Pr(X_{\mathcal{S}_k} = s_{0:k}) = \prod_{n=1}^{k} P_{s_{n-1}s_n} \tag{5.4}$$

In fact, $X_{\mathcal{S}_k}$ follows a categorical distribution with $|\mathcal{S}_k|$ parameters. Sampling from the sequence distribution requires us to compute its cumulative distribution function, which means that we have to define an ordering of the possible sequences.

Definition 17 (Lexicographical Ordering of Sequences)**.** *Given an ordering of states, we define an ordering of sequences such as $s_{0:k} < s_{0:k}'$ if one of the following holds:*

i. $s_{0:k-1} < s_{0:k-1}'$ *or*

ii. $s_{0:k-1} = s_{0:k-1}'$ *and* $s_k < s_k'$

Therefore, we can calculate the cumulative probabilities for the sequences. Given a uniform random variable $U \sim \mathcal{U}(0,1)$, we can choose directly a sample from the sequence space. The relationship between U and $X_{\mathcal{S}_k}$ is shown in the following equation:

$$Pr(X_{\mathcal{S}_k} = s_{0:k}) = Pr(a_{s_{0:k}} < U \leq b_{s_{0:k}}) \tag{5.5}$$

The term $b_{s_{0:k}}$ is defined as the cumulative probability of the $s_{0:k}$ sequence. In the same way, $a_{s_{0:k}}$ will be the cumulative probability of the sequence that precedes $s_{0:k}$ according to the ordering. More formally:

$$\begin{aligned} a_{s_{0:k}} &= \sum_{s_{0:k}{}' < s_{0:k}} Pr(s_{0:k}{}') \\ b_{s_{0:k}} &= \sum_{s_{0:k}{}' \leq s_{0:k}} Pr(s_{0:k}{}') = a_{s_{0:k}} + Pr(s_{0:k}) \end{aligned} \tag{5.6}$$

Although sampling from the sequence distribution is well-defined, it cannot be practically applied in its current form. The number of possible sequences grows exponentially as k increases, a fact that renders Equations (5.4) and (5.6) computationally expensive. However, we shall show next that it is possible to draw a sample from $U \sim \mathcal{U}(0,1)$ that determines the sequence, and recursively generate the transitions involved. A recursive definition for the cumulative sequence probabilities would be useful for this task. Using Definition 17, the cumulative probability of the sequence that precedes $s_{0:k}$ can also be written recursively as follows:

$$a_{s_{0:k}} = Pr(s_{0:k-1}) \sum_{s_k{}' < s_k} P_{s_{k-1}s_k{}'} + a_{s_{0:k-1}} \tag{5.7}$$

Since the uniformly distributed sample U determines the entire k-length sequence, it follows that it also determines all of the k transitions involved. In the DM however, the sequence of the transitions would have been determined by a sequence of uniform samples $U_1, \ldots, U_k$, where $U_n \sim \mathcal{U}(0,1)$ for $1 \leq n \leq k$. Thus, the sequence $U_1, \ldots, U_k$ is equivalent to the sample U used for the state sequence distribution. We shall next define the last sample U_k in terms of U, which gives rise to the following theorem:

Theorem 4. *If $U \sim \mathcal{U}(0,1)$ is used to draw a state sequence sample $s_{0:k}$, then s_k is determined by:*

$$U_k = \frac{U - a_{s_{0:k-1}}}{b_{s_{0:k-1}} - a_{s_{0:k-1}}} \tag{5.8}$$

Proof. We have to show that $a_{s_k} < U_k \leq b_{s_k}$, which means that U_k will select the state s_k, according to Equation (5.1). Since $s_{0:k}$ was selected, Equation (5.5) implies:

$$\begin{aligned} a_{s_{0:k}} < U \leq b_{s_{0:k}} \Leftrightarrow \\ &U > Pr(s_{0:k-1}) \sum_{s_k{}' < s_k} P_{s_{k-1}s_k{}'} + a_{s_{0:k-1}} \\ \text{and} \quad &U \leq Pr(s_{0:k-1}) \sum_{s_k{}' < s_k} P_{s_{k-1}s_k{}'} + a_{s_{0:k-1}} + Pr(s_{0:k}) \end{aligned}$$

We subtract $a_{s_{0:k-1}}$ from all terms, and divide everything by $Pr(s_{0:k-1})$:

$$Pr(s_{0:k-1}) \sum_{s_k{}' < s_k} P_{s_{k-1}s_k{}'} < U - a_{s_{0:k-1}} \leq Pr(s_{0:k-1}) \sum_{s_k{}' < s_k} P_{s_{k-1}s_k{}'} + Pr(s_{0:k})$$
$$\sum_{s_k{}' < s_k} P_{s_{k-1}s_k{}'} < \frac{U - a_{s_{0:k-1}}}{Pr(s_{0:k-1})} \leq \sum_{s_k{}' < s_k} P_{s_{k-1}s_k{}'} + \frac{Pr(s_{0:k})}{Pr(s_{0:k-1})}$$

We substitute $\frac{Pr(s_{0:k})}{Pr(s_{0:k-1})}$ with $P_{s_{k-1}s_k}$ and $Pr(s_{0:k-1})$ with $b_{s_{0:k-1}} - a_{s_{0:k-1}}$:

$$\sum_{s_k{}' < s_k} P_{s_{k-1}s_k{}'} < \frac{U - a_{s_{0:k-1}}}{b_{s_{0:k-1}} - a_{s_{0:k-1}}} \leq \sum_{s_k{}' < s_k} P_{s_{k-1}s_k{}'} + P_{s_{k-1}s_k}$$
$$\sum_{s_k{}' < s_k} P_{s_{k-1}s_k{}'} < \frac{U - a_{s_{0:k-1}}}{b_{s_{0:k-1}} - a_{s_{0:k-1}}} \leq \sum_{s_k{}' \leq s_k} P_{s_{k-1}s_k{}'}$$

which yields:

$$a_{s_k} < U_k \leq b_{s_k}$$

□

Theorem 4 can be used to calculate any of the U_n samples that determine the transitions by simply considering $k = n$, with $n > 1$. For the special case where $k = 1$, the sequence probabilities will be equal to the transition probabilities of the first step. We could then calculate the uniform sample U_{k+1} needed for the next step and recursively update $a_{s_{0:k+1}}$ and $b_{s_{0:k+1}}$ to get the new cumulative sequence probabilities using Equation (5.7). This strategy might not be optimal though, as it requires keeping track of two cumulative probabilities. A cleaner and more efficient solution would be to write U_k in terms of the previous uniform sample U_{k-1}.

Theorem 5. *Assume that U_k and U_{k-1} are related to $U \sim \mathcal{U}(0,1)$ as in Equation (5.8). If U_{k-1} is used to draw a state sample s_{k-1}, then s_k is determined by*

$$U_k = \frac{U_{k-1} - a_{s_{k-1}}}{b_{s_{k-1}} - a_{s_{k-1}}} \tag{5.9}$$

Proof. Given a uniform sample U that determines the sequence, the samples U_k and U_{k-1} can be written as specified in (5.8). If we solve w.r.t. U in both cases, we obtain the following equality:

$$U_k Pr(s_{0:k-1}) + a_{s_{0:k-1}} = U_{k-1} Pr(s_{0:k-2}) + a_{s_{0:k-2}}$$

which yields:

$$U_k = \frac{U_{k-1} Pr(s_{0:k-2})}{Pr(s_{0:k-1})} - \frac{a_{s_{0:k-1}} - a_{s_{0:k-2}}}{Pr(s_{0:k-1})} \tag{5.10}$$

Using (5.7), the numerator of the second fraction above can be written as:

$$a_{s_{0:k-1}} - a_{s_{0:k-2}} = Pr(s_{0:k-2}) a_{s_{k-1}} + a_{s_{0:k-2}} - Pr(s_{0:k-3}) a_{s_{k-2}} - a_{s_{0:k-3}}$$

We also know that $a_{s_{0:k-2}} = Pr(s_{0:k-3}) a_{s_{k-2}} + a_{s_{0:k-3}}$ because of (5.7), so we can rewrite (5.10) as:

$$U_k = \frac{U_{k-1} Pr(s_{0:k-2})}{Pr(s_{0:k-1})} - \frac{a_{s_{k-1}} Pr(s_{0:k-2})}{Pr(s_{0:k-1})}$$

According to the definition of sequence probabilities in (5.4), we have $Pr(s_{0:k-1}) = Pr(s_{0:k-2}) P_{s_{k-2}s_{k-1}}$, which implies:

$$U_k = \frac{U_{k-1} - a_{s_{k-1}}}{P_{s_{k-2}s_{k-1}}}$$

Finally, we can write $P_{s_{k-2}s_{k-1}}$ as a difference of cumulative probabilities to obtain Equation (5.9). □

Starting from some initial transition, we can recursively generate an entire sequence of random samples that are uniformly distributed between 0 and 1. If the previous step utilised a sample $U_{k-1} \sim \mathcal{U}(0,1)$, we know that $a_{k-1} < U_{k-1} \leq b_{k-1}$. If we define U_k according to (5.9), it is easy to show that $0 < U_k \leq 1$, which means that $U_k \sim \mathcal{U}(0,1)$. Although this sequence is produced deterministically, we have shown that it corresponds to the uniform sample needed to sample from the sequence distribution.

Thus, assuming that the quantities $a_{s_{k-1}}$ and $b_{s_{k-1}} - a_{s_{k-1}}$ have to be calculated anyway, generating a sample at each step requires a subtraction followed by a division, as Equation (5.9) implies. This procedure is more efficient than most of the random generator algorithms, in particular the ones that produce high quality random numbers.

5.2.3 Sampling from the Time Distribution

5.2.3.1 Time Discretisation

If we select some particular sequence $s_{0:k}$, the duration of the total of the transitions involved is represented by a $L_{s_{0:k}}$ random variable. In the case of CTMCs this will be the sum of k exponentially distributed independent random variables that determine the duration of each transition, or more formally:

$$L_{s_{0:k}} = \sum_{i=0}^{k} L_{s_i} \tag{5.11}$$

where $L_{s_i} \sim Exp(\lambda_{s_i})$. Therefore, $L_{s_{0:k}}$ will follow hypo-exponential distribution with k parameters, or equivalently $L_{s_{0:k}} \sim Hypo(\lambda_{s_0}, \ldots, \lambda_{s_k})$. To sample directly from $L_{s_{0:k}}$ is only feasible for special cases such as the Erlang distribution, which is a hypo-exponential with k similar parameters. It is possible to transform $L_{s_{0:k}}$ to an Erlang distributed variable by applying uniformisation [54]. This approach is problematic though, as the probability matrix of the embedded DTMC will contain self-loops, in contrast to the original jump chain as defined in (2.6). This means that the uniformised CTMC will involve a larger number of events, a fact that could actually slow the simulation down.

Our attempt of sampling from the hypo-exponential $L_{s_{0:k}}$ efficiently will focus on approximating the exponentially distributed sojourn times with a discrete random variable. The use of a discrete distribution implies that we divide time into intervals, since it involves discrete time-steps rather than continuous. Thus, while a continuous distribution indicates the probability of a transition happening up to a time t, a discrete one indicates the transition probability up to the n-th interval.

The geometric distribution is the most appropriate choice for the task, since it is the discrete analogue of the exponential. Given some exponential random variable $L \sim Exp(\lambda)$, this can be approximated by a geometrically distributed $Y \sim G(p)$ that denotes the number of Bernoulli trials needed to fire a transition with probability p. The geometric distribution is supported in $\mathbb{N}$ excluding zero. Given the length of intervals l, we can map a geometric random variable to $\mathbb{R}^+$ by considering that it is supported in $\{1l, 2l, \ldots\}$. Since Y is geometric, its expected value will be $1/p$ intervals, or l/p in terms of time units. If we make L and Y correspond to the same expected value, that is $1/\lambda = l/p$, it is easy to show that the interval length will be:

$$l = \frac{p}{\lambda} \tag{5.12}$$

Therefore, to determine the amount of time spent in state s_k will involve two steps:

1. Sample from $Y_{s_k} \sim G(p)$. Using a uniform sample $U \sim \mathcal{U}(0,1)$, we choose a $n \in \mathbb{N}^*$ with probability:

$$Pr(Y_{s_k} = n) = Pr(a_{Y_{s_k}} < U \leq b_{Y_{s_k}}) \tag{5.13}$$

   where $b_{Y_{s_k}} = Pr(Y_{s_k} \leq n)$ and $a_{Y_{s_k}} = Pr(Y_{s_k} \leq n-1)$.

2. Calculate the time spent in state s_k:

$$t_{s_k} = nl_{s_k} = n\frac{p}{\lambda_{s_k}} \tag{5.14}$$

The advantage of time discretisation is that we can use the sequence sampling technique presented in Section 5.2.2, and therefore reduce the random samples generated. To illustrate how this is possible, let us consider the stochastic process $\{Y_{s_k,k}\}$ that denotes the collection of geometrically distributed random variables used to approximate the sojourn times in some CTMC. The time index k is discrete, as it denotes the number of jumps in the corresponding CTMC. If we set the same parameter p for those random variables, then they will be independent and identically distributed. We can easily verify that $\{Y_{s_k,k}\}$ is essentially a DTMC, which means that it is possible to generate an entire state sequence using a single uniform sample, as demonstrated in the previous section. The time discretisation was necessary, otherwise it would not be possible to define the discrete-state Markov process $\{Y_{s_k,k}\}$, and therefore employ the trajectory sampling technique.

One desirable property of our approach is that it gives an estimation for the duration of all of the transitions involved in a trajectory. On the contrary, the Gamma sampling used in K-skip only produces the total duration of k transitions. While both approaches use a single random number to determine the duration of trajectories, K-skip is expected to be superior from a performance point of view. However, our method produces trajectories that are as detailed as the ones of the original Markov chain.

One other strength of our approach is that its applicability does not rely on particular model properties. The Gamma sampling used in K-skip assumes that exit rates do not change much during the k steps. This assumption, which is similar to the leap condition in τ-leaping methods, may not hold for some models meaning that it could be an extra source of error. Our method is not exact however, due to the time discretisation employed. The quality of this approximation is discussed in the rest of this section.

5.2.3.2 Quality of Approximation

Since the interval length l is dependent on the parameter p of the Geometric distribution, it is rather intuitive that smaller values for p result in better approximation, as l also tends to get smaller. We are going to characterise the quality of this approximation in a rigorous manner.

Theorem 6. *Let us consider a stochastic process that approximates some CTMC featuring the same state-space S, the same transition probability matrix P, and the same initial distribution $\boldsymbol{\pi}_0$. The approximate process is only different in the sense that the sojourn times are determined by $Y_{s_k} \sim G(p)$, as described in (5.13) and (5.14). Then, the approximate process converges to the corresponding CTMC, as $p \to 0$.*

Proof. Let us define $P(t)$ as the transition probability matrix of a CTMC after time t. Given an initial state distribution vector $\boldsymbol{\pi}_0$, the distribution vector of the CTMC at time t will be:

$$\boldsymbol{\pi}_t = \boldsymbol{\pi}_0 P(t) \tag{5.15}$$

$P(t)$ can be calculated as a weighted sum of different powers of the probability matrix P of the underlying jump chain. The state distribution at t can then be rewritten as follows:

$$\boldsymbol{\pi}_t = \boldsymbol{\pi}_0 \sum_{k=0}^{\infty} P^k \times Pr(k \text{ steps until } t) \tag{5.16}$$

The modified stochastic process that resulted from this geometric approximation will have the same underlying jump chain as the original CTMC. The only term in (5.16) that is different in those two kinds of processes is the probability of k transitions happening until time t. This probability can be expressed as a sum of the probabilities of all sequences with duration less than or equal to t, weighted by the sequence probabilities:

$$Pr(k \text{ steps until } t) = \sum_{s_{0:k} \in \mathcal{S}_k} Pr(L_{s_{0:k}} \leq t) Pr(s_{0:k}) \tag{5.17}$$

In order to show that the behaviour of some CTMC as given in (5.16) tends to be more accurately approximated by the modified process as $p \to 0$, it is sufficient (although not necessary) to show that $Pr(k \text{ steps until } t)$ tends to be the same for the two kinds of processes as $p \to 0$. The modified process will have same sequence probabilities as its corresponding CTMC, since the jump process is the same. Thus, two corresponding processes are only different w.r.t the distribution of $L_{s_{0:k}}$. Therefore, it is sufficient (although not necessary again) to show that the cumulative distribution functions for the sojourn times tend to be the same as $p \to 0$. Since the true sojourn time L_{s_k} is approximated by lY_{s_k}, we have to calculate the following limit:

$$\begin{aligned}\lim_{p\to 0} Pr(lY_{s_k} \le t) &= \lim_{p\to 0} Pr(Y_{s_k} \le t/l) \\ &= \lim_{p\to 0} 1-(1-p)^{t/l}\end{aligned}$$

We can substitute the interval length l on the exponent according to (5.12).

$$\begin{aligned}\lim_{p\to 0} Pr(lY_{s_k} \le t) &= \lim_{p\to 0} 1-(1-p)^{\lambda t/p} \\ &= 1-(\lim_{p\to 0}(1-p)^{1/p})^{\lambda t} \\ &= 1-e^{-\lambda t} = Pr(L_{s_k} \le t)\end{aligned}$$

□

The p parameter is a probability, so we have $0 < p \le 1$. Theorem 6 implies that the smaller the value of p is, the better the approximation will be. However, a value for p that is too small can make the geometric sampling described by (5.13) inefficient, as the cumulative probabilities of the form $Pr(Y_{s_k} \le n)$ will involve too many terms. Hence, we need a trade-off between approximation quality and efficiency. In the experiments that follow, we use two different values: $p = 1$ that implies deterministic time-steps that depend on the current state only, and $p = 0.1$ which we think that it is a more appropriate choice, judging by the experimental results of Section 5.3.

5.2.4 Implementation Issues

Although sampling from the sequence distribution as discussed in Section 5.2.2 is theoretically correct, it gives rise to some computational issues. In most computer systems, the mantissa for the double-precision floating-point format contains 53 bits. That is why most random generators produce doubles of the form: $m \times 2^{-53}$, where m is a uniformly distributed integer. In other words, a random generator is capable of producing 2^{53} different values. The TSS algorithm will have 2^{53} different inputs resulting in 2^{53} different trajectories at most. The number of the possible trajectories can easily exceed this value even for not so long simulation runs, since it grows exponentially with the number of simulation events. Therefore, it is inevitable that we will miss a significant number of possible state sequences.

This effect can be eliminated if we sample trajectories of some particular length k, such that the number of possible sequences are significantly smaller than the number of uniformly distributed doubles. A value $k = 10$ is a reasonable choice that suits most of the models that we have encountered in practice. Given 53-bits of precision for the mantissa, we have $2^{53} \approx 9 \times 10^{15}$ different possible random numbers. The largest model that we have tested is LacY [56] involving 21 bio-chemical reactions, which means that the maximum number of transitions available is also 21. If we set $k = 10$, we have $21^k \approx 1.66 \times 10^{13} \ll 2^{53}$.

Each step in TSS consists of two actions: state sequence sampling as described in Section 5.2.2, and time sampling using geometric approximation. These concepts are summarised in Algorithm 4, which involves two parameters: p that controls the granularity of the geometric distribution and the length k of the trajectories to sample. In our implementation the probabilities of the geometric distribution have been pre-calculated for efficiency.

Algorithm 4 Trajectory Sampling Simulation

```
1:  Initialise system state and set $0 < p \leq 1$ and $k \geq 1$
2:  Draw samples $U_L \sim \mathcal{U}(0,1)$ and $U_X \sim \mathcal{U}(0,1)$
3:  **while** $t < t_{final}$ **do**
4:      Given that $M$ transitions are currently possible, calculate the transition rates $\lambda_m$,
        $\forall m \in \{1, 2, \ldots, M\}$
5:      Calculate $\lambda = \sum_{m=1}^{M} \lambda_m$, which is the rate of leaving the current state
6:      Using sample $U_L$, draw $n$ from the geometric distribution $G(p)$
7:      Using sample $U_X$, pick transition $m$ with probability $\lambda_m/\lambda$
8:      Update time: $t \leftarrow t + np/\lambda$
9:      Update state with effect of transition $m$
10:     **if** iteration mod $k \neq 0$ **then**
11:         Update $U_L$ and $U_X$ according to Equation (5.9)
12:     **else**
13:         Draw samples $U_L \sim \mathcal{U}(0,1)$ and $U_X \sim \mathcal{U}(0,1)$
14:     **end if**
15: **end while**
```

5.3 Experiments

In this section, we present a series of experiments to evaluate the efficiency and the accuracy of our method. We want to investigate whether TSS delivers improved performance with respect to the DM. Two different parameter values are used for the geometric approximation in TSS: $p = 1$ and $p = 0.1$.

Moreover, we also compare our approach with the K-skip method [9], which also relies on minimising random number generation. The differentiation however is on that the sojourn times are determined by a Gamma distribution with shape parameter k, which approximates the sum of k exponential random variables. The value of k is determined automatically, depending on a user defined error parameter. The error parameter that we have used for K-skip is 0.01, which is the smallest value used in the original work. The assumption is that the exit rates for subsequent states are similar, which is a concept used in many approximate simulation methods, including τ-leaping [41] and *K-leap* [10]. K-skip is expected to be normally more efficient, because of the use of a Gamma distribution. However, we think that the absence of similar assumptions for TSS implies that it can be effective for arbitrary models, in contrast with K-skip.

We have applied our approach to simulate three different models of bio-chemical reaction networks. The first one is the Schlögl model as appeared in [20]. It is a relatively small model featuring 4 reactions and 3 species, it is interesting however because of its bistable distribution. The second model is LacY, which involves 21 reactions and 22 species, as appeared in [56]. The third example is Goldbeter's oscillatory model [43] as presented in [23], which involves 7 reactions and 6 species. A Bio-PEPA specification of the aforementioned models can be found in Appendix A. Both models have been simulated using the ODM, K-skip, and TSS. The implementation of both K-skip and TSS is based on the ODM, hence any efficiency comparisons have ODM as a baseline.

5.3.1 Evaluation of Efficiency

The efficiency of our algorithm stems from the fact that it generates fewer random numbers. Our intention is to demonstrate that minimising the random numbers generated results in improved performance, no matter how efficient the implementation is. One of the most popular random generators in the literature is *Mersenne Twister* (MT) [67]. It produces high quality random numbers while it exhibits performance comparable to the most efficient algorithms of its kind, as can be seen in [62]. We have developed our algorithm in Java using a number of open-source libraries that contain implementations of MT, namely Apache Commons, CERN Colt, JAMES II [51] and

SSJ [63]. The implementations used produce doubles whose mantissa precision is 53 bits.

Table 5.1: Running times in seconds for 10^5 simulation runs

(a) Schlögl model, $t_{final} = 4$

			TSS	
	ODM	K-skip	$p = 1$	$p = 0.1$
Apache Commons	407	458	307	333
CERN Colt	391	484	308	302
JAMES II	393	483	309	318
SSJ	445	555	303	329

(b) LacY model, $t_{final} = 1000$

			TSS	
	ODM	K-skip	$p = 1$	$p = 0.1$
Apache Commons	8759	5588	6930	6951
CERN Colt	9043	5568	6974	6915
JAMES II	9684	5490	7043	6944
SSJ	10452	5562	7248	7322

(c) Goldbeter's model, $t_{final} = 10$

			TSS	
	ODM	K-skip	$p = 1$	$p = 0.1$
Apache Commons	12264	9354	10678	10615
CERN Colt	12685	9514	10662	10658
JAMES II	13531	9857	10598	10596
SSJ	14636	9269	10719	10734

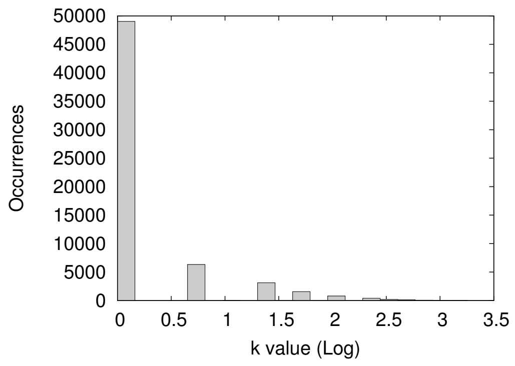
Table 5.1 contains the running times for different random generators. The experiments have been performed in an Intel® Xeon$^{\text{TM}}$ E5410 @ 2.33GHz PC running Scientific Linux 6. The results imply that TSS is about 15 $\sim$ 20% faster than the ODM. A second observation is that using $p = 1$ is not significantly faster than TSS using $p = 0.1$ for the geometric distribution. This means that there is no need to use

a value for p greater than 0.1, as this would not result in a significant improvement in efficiency.

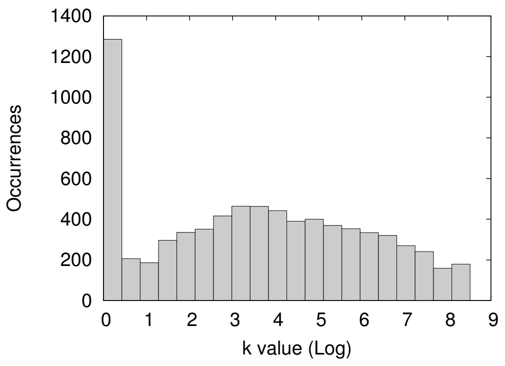
Comparing running times for K-skip and TSS, the results are mixed. K-skip outperforms TSS for the LacY model in Table 5.1(b), and the Goldbeter's model in Table 5.1(c). These results are not surprising, since K-skip determines the total duration of k events by sampling from a Gamma distribution, while our approach determines the duration of every single event happening. We note that the speed-ups observed for K-skip are smaller than the values reported in [9]. This is due to the MT random number generator, which is more efficient than the ran2 algorithm used in Cai & Wen paper, as pointed out in [62]. Because we are using a more efficient random generator there is less scope to deliver speed-ups over the ODM. If we consider this difference, the results we have found for K-skip seem to comply with the ones reported in the original work.

Regarding the Schlögl model in Table 5.1(a), K-skip appears to be less efficient than both TSS and the baseline approach. The effectiveness of K-skip is strongly related to the value of the shape parameter k for the Gamma distribution. At each simulation step, a different k value is selected, which depends on the current simulation state. In order to understand the source of inefficiency, we have to monitor the k values selected by the algorithm. The log-scaled histograms of Figure 5.1 capture the distribution of k for the three models considered. The histograms are presented in a logarithmic scale in order to maximise visibility, as the distribution of k is heavy-tailed.

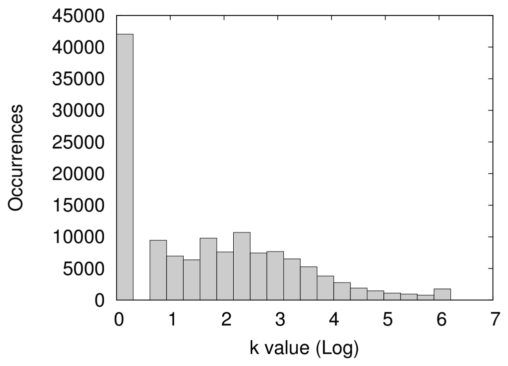
We can see that k typically takes large values for both the LacY model in Figure 5.1(b) and the Goldbeter's model in Figure 5.1(c). However, the selected k values for the Schlögl model in Figure 5.1(a) are very small, while the majority of the observations are $k = 1$. Hence, the computational overhead of selecting k outweighs any efficiency improvement that comes with K-skip. Those differences in the selected k values can be attributed to the fact that the Schlögl model does not comply with the assumptions that K-skip relies on. The K-skip method responded by selecting small values for k, a fact that had a detrimental effect on performance. In contrast, since TSS does not rely on similar assumptions, it has been consistently efficient for all the models tested.



(a) Schlögl model



(b) LacY model



(c) Goldbeter's model

Figure 5.1: Distribution of the logarithm of k values used in 10 simulation runs of the K-skip method

5.3.2 Evaluation of Accuracy

A second issue that has to be explored is whether the stochastic process described by Algorithm 4 is equivalent to the original Markov chain. The convergence is ensured as $p \to 0$ when $k = 1$. The simulation will be still exact even if $k > 1$ as implied by Theorems 4 and 5. However, the use of the geometric approximation means that we have a slightly altered process that approximates the original. To assess the quality of this approximation we construct the histograms for various rewards (i.e. species populations) in the models used, as it would have been impractical to compare the entire state-space distribution for models of that size. We then calculate the *histogram distance* [20], which is the euclidean distance between the histograms of the true and the approximate distribution.

It is important to note that the histogram distance will always be larger than zero, even if the simulation is exact, since the empirical distributions which result from simulation are always going to be different. In order to determine whether the distance calculated is significant, it has to be compared with the corresponding self-distance. The histogram self-distance depends on the number of samples drawn and the number of histogram intervals used. A value for the histogram distance that is smaller than the self-distance implies that the two distributions are practically identical for the given number of samples. According to [20], an upper bound for the average histogram self-distance given N samples is independent from the distribution and it can be calculated using $\sqrt{(4K)/(\pi N)}$, where K is the number of intervals in the histogram. For the examples that follow, we have considered $K = 50$.

Table 5.2 summarises the histogram distances for several species populations and time-points in the models considered. For TSS with $p = 1$, some of the distances are slightly larger than the self-distance (the values written in italics). This implies that we have a reasonably good approximation but the error introduced by using fixed times is observable for the number of samples considered. However, the approximation quality is better when using TSS with $p = 0.1$, as it was expected. The histogram distance from the true distribution is at the same level or smaller than the self-distance estimated almost in all cases. This means that the error observed is within the limits of the error inherently introduced by the simulation process. Those findings support the claim that TSS with parameter $p = 0.1$ for the geometric approximation is an accelerated simulation approach that is almost exact.

While K-skip has been more efficient than our approach for the LacY and Goldbeter models, Table 5.2 suggests that it is not as accurate in some cases. Most of the histogram distances for the Schlögl and the LacY model are greater than either the self-distance or the corresponding distances calculated for both versions of TSS. It seems that the assumption that the rates of subsequent states are similar might introduce some errors, a fact that renders K-skip less appropriate for some models. Our approach generalises to systems where this assumption is not valid. Moreover, our use of the geometric approximation specifies the duration of every single event happening, which can be important for some systems.

Table 5.2: Histogram distances for 10^6 simulation runs (self-distance: 0.0080)

(a) Schlögl model

	K-skip			TSS ($p = 1$)			TSS ($p = 0.1$)		
Time	X	B1	B2	X	B1	B2	X	B1	B2
1	*0.0082*	0.0047	0.0060	0.0061	0.0054	0.0060	0.0052	0.0068	0.0052
2	*0.0172*	0.0079	0.0052	0.0058	0.0073	0.0062	0.0060	0.0058	0.0050
3	*0.0156*	0.0059	0.0059	0.0072	0.0046	0.0067	0.0075	0.0062	0.0056
4	*0.0122*	0.0066	0.0074	0.0070	0.0056	0.0065	0.0054	0.0049	0.0052

(b) LacY model

	K-skip			TSS ($p = 1$)			TSS ($p = 0.1$)		
Time	lactose	PLac	product	lactose	PLac	product	lactose	PLac	product
250	0.0070	*0.0090*	0.0064	0.0062	0.0005	0.0071	0.0045	0.0011	0.0054
500	0.0040	0.0074	*0.0087*	0.0042	0.0011	*0.0083*	0.0030	0.0004	0.0071
750	0.0041	0.0077	0.0074	0.0034	0.0004	*0.0086*	0.0050	0.0001	0.0076
1000	0.0044	*0.0086*	*0.0081*	0.0040	0.0004	*0.0087*	0.0032	0.0002	0.0079

(c) Goldbeter's model

	K-skip			TSS ($p = 1$)			TSS ($p = 0.1$)		
Time	active_M	active_X	C	active_M	active_X	C	active_M	active_X	C
2.5	0.0065	0.0068	0.0074	0.0071	0.0039	0.0039	0.0048	0.0040	0.0036
5.0	0.0067	0.0059	0.0055	0.0067	0.0066	0.0054	0.0066	0.0053	0.0052
7.5	0.0070	*0.0088*	0.0037	0.0068	*0.0082*	0.0054	0.0080	0.0079	0.0060
10.0	0.0071	0.0063	0.0067	0.0055	0.0048	*0.0081*	0.0041	0.0061	0.0062

5.4 Summary

Trajectory sampling simulation requires fewer random samples to generate Markov chain trajectories. This is achieved by using a single random number to determine an entire sequence of transitions. We have proven that the random number required to select the next transition can be written in terms of the random number that selected the previous transition. This leads to a recursive update of a single random number that determines an entire state sequence. In the case of CTMCs a second random number is used to determine the length of this sequence. The same concept has been used by approximating the exponentially distributed times with a geometric distribution with parameter p that controls the quality of this approximation.

We have simulated three biochemical models of different nature to assess the efficiency and the accuracy of the our method. The experimental results show that our approach is about $15 \sim 20\%$ faster than the ODM, while the errors observed were found to be negligible. K-skip method I, which is a similar approach, was found to be more efficient but less accurate in most of the cases. Moreover, we have seen that K-skip relies on certain assumptions with respect to the model; if the model in question is not compatible with those assumptions, then K-skip can be problematic. Thus, TSS can be thought of as an alternative to K-skip in cases where this is possibly inappropriate.

There are also some practical considerations with respect to the length k for the trajectories to be sampled. A too large value for k might result in missing possible state sequences, while a value too small will degenerate trajectory sampling simulation to the ODM. We have used $k = 10$ for the experiments produced, but in the case of larger models we would have to set a smaller value for k. We think that $k = 5$ is appropriate even for very large models. For example, given a model with 500 reactions we have: $500^k \approx 3.125 \times 10^{13} \ll 2^{53}$.

Chapter 6

Case Study on Cloud Computing

In this chapter, we provide a unifying view of the concepts that we have introduced in this thesis. We devise a case study in order to further discuss the methodologies proposed over a realistic problem. The area of interest is cloud computing services, which provide many challenges from an engineering point of view, as systems and infrastructures have to be able to scale in order to support increasing demand.

Cloud systems typically consist of many inhomogeneous components, whose interactions define a complex behaviour. Performance modelling can be an effective way to derive expectations for such systems. The PEPA language in particular offers a framework to represent parts of a system in isolation, while larger components are formed as compositions of simpler building blocks. It is our position that this systematic approach to modelling simplifies and rationalises the modelling process. However, increased complexity often renders analysis difficult, which makes it an ideal example to demonstrate the potential of approximation techniques.

The problem under investigation is the scalability of different routing policies in a *Platform as a Service* (PaaS) system. In short, a routing algorithm is responsible for directing the workload to a number of leased servers. Quantitative modelling and analysis can provide valuable insights into the effect of different routing policies. This process essentially involves experimentation with different workloads for different system sizes. An approximate method allow us to efficiently deal with complex systems, but it naturally distorts part of the original behaviour. The objective of this case study is to demonstrate what kind of questions can be answered when applying our approaches, and most importantly, how the approximate results obtained may affect the conclusions that can be drawn.

In Section 6.1, we give a small introduction to cloud services, and PaaS in particular. Moreover, we discuss the routing issue that has been raised for the Heroku PaaS provider, which has been the inspiration for our case study. In Section 6.2, we discuss two models for routing policies for Heroku. Comparative results for the policies considered are presented in Section 6.3.

6.1 Cloud Services

Cloud computing is a term that describes the access to distributed hardware or software resources that are available as a service on demand, typically over the Internet. Cloud solutions provided typically follow one of the fundamental service models [37]:

Infrastructure as a Service (IaaS) refers to access to computing resources, including actual hardware or virtualised computers, storage, bandwidth or other resources. For example, that could involve renting machines to run specialised jobs or some particular applications.

Platform as a Service (PaaS) builds upon IaaS and further provides the clients with a customised solution stack. This includes operating systems, programming languages, libraries, web servers, databases and software tools that developers can make use of to create services.

Software as a Service (SaaS) depends on both IaaS and PaaS, as shown in Figure 6.1, and provides access to remote software applications in a manner completely transparent to the end user.

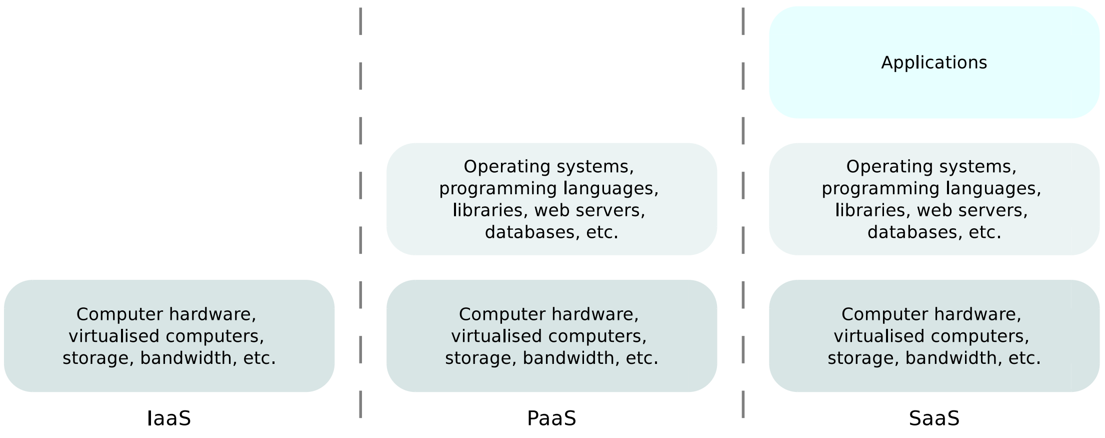

Applications
Operating systems, programming languages, libraries, web servers, databases, etc.
Operating systems, programming languages, libraries, web servers, databases, etc.
Computer hardware, virtualised computers, storage, bandwidth, etc.
Computer hardware, virtualised computers, storage, bandwidth, etc.
Computer hardware, virtualised computers, storage, bandwidth, etc.
IaaS
PaaS
SaaS

Figure 6.1: Service models of cloud computing

The term “cloud computing” characterises the business model under which such services are made available, rather than the technologies that make it happen, including the web or the grid. What the end users normally think of as the cloud is described by SaaS, as they only interact with software applications without knowledge or direct interference with the computer resources involved. PaaS and IaaS describe the cloud from a developer's point of view. PaaS in particular, includes all the necessary tools and equipment to deliver web and cloud services.

The traditional way of developing and deploying a remote application involves a dedicated team of developers that make use of frameworks such as J2EE and .NET, in order to configure network, database and hardware resources. This means that a significant part of the workforce is allocated to simply maintaining the platform they are working on. From that respect, PaaS tends to change the way on-demand applications are delivered. Having the entire solution stack provided as a service can have significant benefits to web service and SaaS providers, as they can focus on the development of services, rather than on the underlying infrastructure. In this way, they seamlessly use a constantly updating platform which has been tailored to their needs. The maintenance effort is moved towards the PaaS provider, who is responsible for issues regarding scalability and fault tolerance, enabling companies to focus on the business logic of their applications.

In order to make our discussion on PaaS less abstract, we shall focus on a particular example in the rest of this section, so as to highlight possible issues, and illustrate how performance modelling may contribute in identifying such issues and support design decisions.

6.1.1 The Heroku Case

Heroku as a PaaS provider offers an integrated framework that enables developers to deploy and support web-based applications. Several programming languages are supported, including Ruby, Node.js, Clojure, Java, Python, and Scala. Clients are supposed to upload the source code for their application, together with a file that describes the software dependencies. The Heroku platform will then be able to build the application, which will be executed on one or more virtualised machines, which are known as *dynos*.

According to the on-line Heroku specification documents[1], a dyno is a lightweight environment running a single command at a time. This functionality is implemented by an isolated virtualised server running Ubuntu 10.04 or Debian Lenny 5.0. Dynos

[1]https://devcenter.heroku.com/

are claimed to provide a secure and performance-consistent environment to run an application. There are two kinds of dynos available: *web dynos* which respond to HTTP requests, and *worker dynos* which execute background jobs. Commands are not supposed to be interrupted, thus concurrency is achieved by employing more than one dynos. Increasing the number of web dynos will increase the concurrency of HTTP requests, while more worker dynos provide more capacity for processes running in the background. Therefore, all the client has to do is to upload the source code of the application and scale it to a number of dynos. The idea is that once a service request appears, Heroku will be responsible for assigning that request to one of the dynos that have been leased by the client, by following a routing policy as outlined in Figure 6.2.

The routing policy is the key component that we shall investigate in this case study. We list below two routing policies that have historically been used by Heroku:

Random Routing implies that a new request is directed to a randomly-selected web dyno. The premise of random routing is that the load is balanced across the dynos in the long term.

Smart Routing involves tracking the availability of each dyno and the load is directed accordingly, thus minimising the number of dynos that remain idle.

Although explicit information on the implementation of these policies is not available, it is straightforward to model the desired behaviour for each policy at a high-level.

The next important business choice for the client is to determine how many dynos should be leased. Of course, this depends on the workload. Naturally, the heavier the workload is, the more dynos will be needed. In the ideal case, every service should be tailed to the needs of the corresponding client. Typically, clients may have a rough idea of the expected workload. However, they might find it very difficult to accurately estimate the number of the machines needed, as this is also dependent on the internal architecture of the PaaS system, which is supposed to be transparent. This kind of information has to be provided by the PaaS provider, among other tools (such as monitoring tools) that can contribute to business decisions for the client.

Performance modelling is a natural way to produce such estimates in a rigorous manner. Despite the fact that modelling relies on rather strong assumptions, if done appropriately it can provide us with a very useful insight into some even not so apparent aspects of a system. A failure to produce expectations about a system's performance may result in unexpected delays that will inevitably affect the quality of service for the

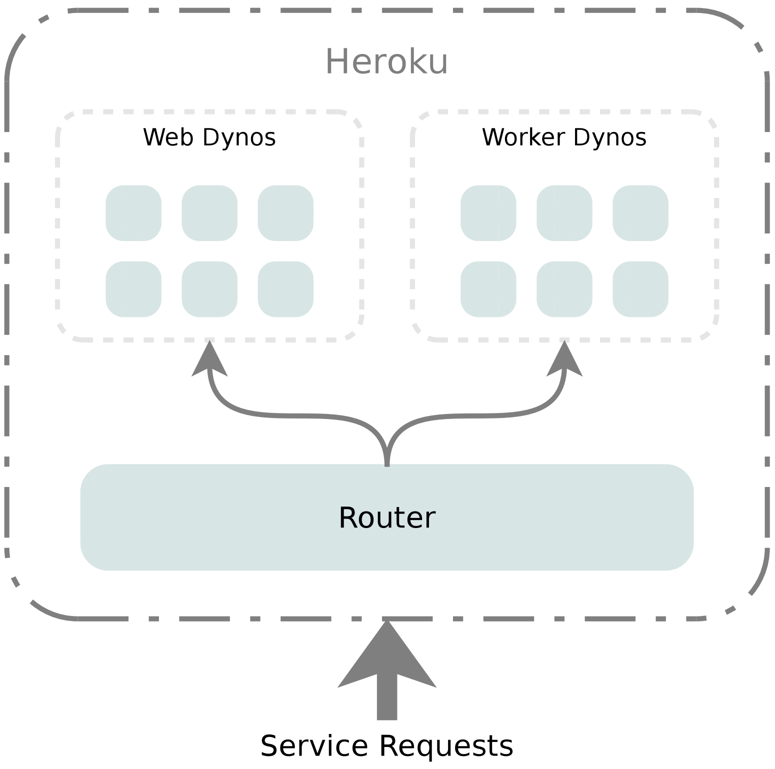


Figure 6.2: The basic structure of Heroku

client. Such an outcome is not improbable at all; in fact, we shall report a particular incident which has inspired the current case study.

Rap Genius[2] is a website that aims to provide a critical and artistic insight into the lyrics of rap songs. The cultural contribution of Rap Genius is remarkable, however, in terms of the current poetically sterile thesis, we shall focus on some technical aspects only. The website users have access to content via HTTP requests, and they are able to add annotations to content. Rap Genius makes this service available via the cloud, and Heroku in particular.

In the beginning of 2013, Rap Genius reported unusually long average response times, despite the large number of dynos leased by the website[3]. The average response time reported by the Heroku platform was as low as 40 ms, while the response time experienced by the users has been 6330 ms. This difference has been attributed to the fact that the requests are waiting in the local queues of the dynos. Therefore, given that the actual service has not been any slower than usual, this could suggest that the system has simply been overloaded. Nevertheless, according to Rap genius, there has not been

[2]http://rapgenius.com/
[3]http://rapgenius.com/James-somers-herokus-ugly-secret-lyrics

any significant change in the workload, which has been as high as 9000 requests per minute. Eventually, this considerable increase in the response time has been attributed to the fact that the Heroku routing policy has been changed from smart to random.

Our objective is not to assess the quality of service provided by Heroku, or recreate absolutely realistic expectations of the response time for Rap Genius. Instead, we want to demonstrate how modelling with Markov chains may capture the effect of different routing policies, and most importantly, whether the approximation methods that we propose throughout this thesis produce adequately accurate results to allow modellers to reach the same conclusions in a more efficient way.

6.2 Modelling Heroku Routing with PEPA

Before modelling the routing policies, we shall describe the basic components and the interactions among them in an abstract way. As shown in Figure 6.3, the model we consider involves two classes of dynos, web and worker, and a router component.

A Poisson process governs the arrivals of web requests, which are initially directed to the router. The router component is responsible for forwarding a request to a web dyno. When a web dyno receives a request, there are two possibilities: it can either service the request or create a new request that can be serviced by a worker dyno. In that case, the current job will simply migrate from a web to a worker dyno, and the router is still responsible for redirecting the request accordingly. It is assumed that a small fraction of the requests are migrated; more specifically, we consider a migration probability equal to $1/9$. This does not imply however that we split the Poisson arrivals into two Poisson processes, as the job is supposed to spend some time in the web dyno before migrating. This is the only way a worker dyno may be accessed, as the users are assumed to produce HTTP requests only. The generation of a worker request captures the possibility that a job may require some background computation process.

We shall identify some activity types that need to be present in our model, regardless of the routing policy. These activities will be associated with exponentially distributed durations. Table 6.1 summarises the rates of the events considered. The number of requests will be governed by a Poison process with rate $r_{request}$. The request arrival rate $r_{request}$ will control the assumed workload in the system. It is actually the variable we are going to experiment with, so it will take values within a range from 40 to 150 sec^{-1}, which corresponds to 9000 requests per minute.

Regarding the service rate, this is going to be dependent on the type of dyno. In

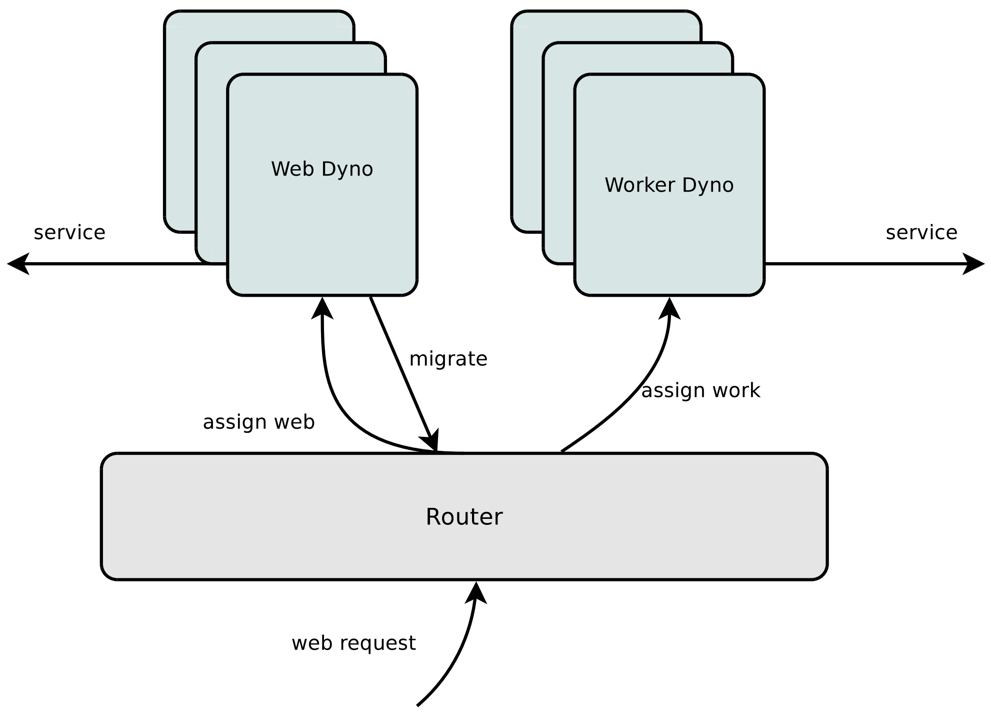


Figure 6.3: The Heroku configuration considered

Table 6.1: The rate values used in the examples

Variable Name	Value (sec^{-1})
$r_{request}$	[40, 60, 150]
r_{web}	8
$r_{migrate}$	1
r_{worker}	4
$r_{response}$	20
r_{assign}	500

both cases, we assume that service is broken down in two parts: the actual service and the response. The actual service part covers the amount of work that a dyno needs to produce a result. The service time depends on the type of the job. While both types of dynos are identical with respect to their computational capabilities, the worker dynos deal with more demanding tasks, which is reflected in a lower service rate. Therefore the average web service time is $1/r_{web} = 0.125$ sec, while for the worker dyno services we have an average time of $1/r_{worker} = 0.25$ sec. The response part represents the time needed by a dyno to transmit the results to the user. It is considered to be identical in both cases, as it only depends on the network. Moreover, response takes place at a considerably higher rate than the actual service, so it has rate $r_{response} = 20$.

As we shall see in the PEPA models in the rest of this section, it is assumed that there is a race condition between migration and web service. Thus, the rate of migration will control the migration probability. By considering $r_{migrate} = 1$ and given that we have $r_{web} = 8$, we essentially impose a migration probability equal to $1/9$.

Finally, it is assumed that assignment happens almost instantaneously, as it is an activity that depends on the resources allocated to the routing component only. Given the current state of the system, it is fair to expect that any decision will take place very quickly. This is reflected by the significantly high rate $r_{assign} = 500$, or 0.002 seconds average duration.

In the rest of this section, we present two PEPA models that implement the routing policies in question. We assume that each dyno has its own queue, thus we are interested in observing how the local dyno queues are affected by each policy. We note that a formalism such as queueing networks is also appropriate to investigate such issues. However, the compositional structure of PEPA models allow for efficient model reduction, as discussed in Chapter 4.

6.2.1 Random Routing Policy

A dyno can be anything between idle, occupied or having one or more requests in its local queue. According to the random routing policy, a router is supposed to randomly assign jobs to dynos, regardless of their state. In terms of PEPA models, and subsequently CTMCs, it is very simple to represent such a probabilistic behaviour.

Web dynos are represented by components $WebDyno_i$, where the subscript i denotes the number of requests in the local dyno queue. For $WebDyno_i$, three activities are possible; *service* realises the main web service part, whose completion proceeds to the response stage, carried out by $WebDyno_{ia}$. It is assumed that a response cannot be interrupted, thus no new job can be assigned or enqueued at this point. Given that the response rate is significantly higher than the web service rate (see Table 6.1), this should not affect the availability of the dyno. The *migrate* activity generates a migration request and decreases the queue length. Finally, the $assign_{web}$ activity adds a request to the queue.

$$\begin{aligned}
WebDyno &\stackrel{def}{=} (assign_{web}, \top).WebDyno_0 \\
WebDyno_i &\stackrel{def}{=} (service, r_{web}).WebDyno_{ia} \\
&+ (migrate, r_{migrate}).WebDyno_{i-1} \\
&+ (assign_{web}, \top).WebDyno_{i+1} \\
WebDyno_{ia} &\stackrel{def}{=} (response, r_{response}).WebDyno_{i-1}
\end{aligned}$$

The components $WebDyno_i$ and $WebDyno_{ia}$ represent the two stages of a web service. In both cases, the web dyno is considered to be occupied. The idle state is denoted by *WebDyno*, which only performs an $assign_{web}$ activity.

Regarding the worker dynos, they have a similar but simpler structure, as there is no job migration option in this case. Other than that, assignment has been labelled by a distinct activity $assign_{worker}$, in order to distinguish between the two possible types of assignments.

$$\begin{aligned}
WorkerDyno &\overset{def}{=} (assign_{worker}, \top).WorkerDyno_0 \\
WorkerDyno_i &\overset{def}{=} (service, r_{worker}).WorkerDyno_{ia} \\
&\quad + (assign_{worker}, \top).WorkerDyno_{i+1} \\
WorkerDyno_{ia} &\overset{def}{=} (response, r_{response}).WorkerDyno_{i-1}
\end{aligned}$$

The routing component is characterised by a set of states that denote the number of requests in the router queue. At any state, the router should be able to accept a web request or a migration request, either of which will be added in the router queue. Remember that web requests are modelled by a Poisson process, thus the *request* activity has a constant rate for any state of the router component. When a new job arrives, the router will attempt to direct it to any of the web or worker dynos, depending on the type of the request. It is more convenient to model the router as two queues, one for each type of dyno. So for the web requests we have:

$$\begin{aligned}
WebRouter_0 &\overset{def}{=} (request, r_{request}).WebRouter_1 \\
WebRouter_i &\overset{def}{=} (request, r_{request}).WebRouter_{i+1} \\
&\quad + (assign_{web}, r_{assign}).WebRouter_{i-1} \\
WebRouter_n &\overset{def}{=} (request, r_{request}).WebRouter_n \\
&\quad + (assign_{web}, r_{assign}).WebRouter_{n-1}
\end{aligned}$$

where n denotes the maximum size for the corresponding queue. If the maximum size is reached, it is assumed that any new request will be discarded until the queue is not full. In the same way, for migration requests we have:

$$\begin{aligned}
WorkerRouter_0 &\overset{def}{=} (migrate, \top).WorkerRouter_1 \\
WorkerRouter_i &\overset{def}{=} (migrate, \top).WorkerRouter_{i+1} \\
&\quad + (assign_{worker}, r_{assign}).WorkerRouter_{i-1} \\
WorkerRouter_n &\overset{def}{=} (migrate, \top).WorkerRouter_n \\
&\quad + (assign_{worker}, r_{assign}).WorkerRouter_{n-1}
\end{aligned}$$

Finally, the router will be the parallel composition of the components above.

Figure 6.4 outlines the complete model of the random routing policy. Note that the model imposes a maximum dyno queue length equal to 1. The main reason behind this modelling choice is to keep the component state-space at relatively low levels, in order to avoid excessive state-space explosion. As we shall see later in Section 6.3, this model is adequate to observe the qualitative difference between a random and a smart routing policy.

$$
\begin{aligned}
\mathit{WebDyno} &\overset{def}{=} (\mathit{assign}_{web}, \top).\mathit{WebDyno}_0 \\
\mathit{WebDyno}_0 &\overset{def}{=} (\mathit{service}, r_{web}).\mathit{WebDyno}_{0a} + (\mathit{migrate}, r_{migrate}).\mathit{WebDyno} \\
&+ (\mathit{assign}_{web}, \top).\mathit{WebDyno}_1 \\
\mathit{WebDyno}_{0a} &\overset{def}{=} (\mathit{response}, r_{response}).\mathit{WebDyno} \\
\mathit{WebDyno}_1 &\overset{def}{=} (\mathit{service}, r_{web}).\mathit{WebDyno}_{1a} + (\mathit{migrate}, r_{migrate}).\mathit{WebDyno}_0 \\
\mathit{WebDyno}_{1a} &\overset{def}{=} (\mathit{response}, r_{response}).\mathit{WebDyno}_0
\end{aligned}
$$

$$
\begin{aligned}
\mathit{WorkerDyno} &\overset{def}{=} (\mathit{assign}_{worker}, \top).\mathit{WorkerDyno}_0 \\
\mathit{WorkerDyno}_0 &\overset{def}{=} (\mathit{service}, r_{worker}).\mathit{WorkerDyno}_{0a} + (\mathit{assign}_{worker}, \top).\mathit{WorkerDyno}_1 \\
\mathit{WorkerDyno}_{0a} &\overset{def}{=} (\mathit{response}, r_{response}).\mathit{WorkerDyno} \\
\mathit{WorkerDyno}_1 &\overset{def}{=} (\mathit{service}, r_{worker}).\mathit{WorkerDyno}_{1a} \\
\mathit{WorkerDyno}_{1a} &\overset{def}{=} (\mathit{response}, r_{response}).\mathit{WorkerDyno}_0
\end{aligned}
$$

$$
\begin{aligned}
\mathit{WebRouter}_0 &\overset{def}{=} (\mathit{request}, r_{request}).\mathit{WebRouter}_1 \\
\mathit{WebRouter}_1 &\overset{def}{=} (\mathit{request}, r_{request}).\mathit{WebRouter}_2 + (\mathit{assign}_{web}, r_{assign}).\mathit{WebRouter}_0 \\
\mathit{WebRouter}_2 &\overset{def}{=} (\mathit{request}, r_{request}).\mathit{WebRouter}_3 + (\mathit{assign}_{web}, r_{assign}).\mathit{WebRouter}_1 \\
\mathit{WebRouter}_3 &\overset{def}{=} (\mathit{request}, r_{request}).\mathit{WebRouter}_3 + (\mathit{assign}_{web}, r_{assign}).\mathit{WebRouter}_2
\end{aligned}
$$

$$
\begin{aligned}
\mathit{WorkerRouter}_0 &\overset{def}{=} (\mathit{migrate}, \top).\mathit{WorkerRouter}_1 \\
\mathit{WorkerRouter}_1 &\overset{def}{=} (\mathit{migrate}, \top).\mathit{WorkerRouter}_2 + (\mathit{assign}_{worker}, r_{assign}).\mathit{WorkerRouter}_0 \\
\mathit{WorkerRouter}_2 &\overset{def}{=} (\mathit{migrate}, \top).\mathit{WorkerRouter}_3 + (\mathit{assign}_{worker}, r_{assign}).\mathit{WorkerRouter}_1 \\
\mathit{WorkerRouter}_3 &\overset{def}{=} (\mathit{migrate}, \top).\mathit{WorkerRouter}_3 + (\mathit{assign}_{worker}, r_{assign}).\mathit{WorkerRouter}_2
\end{aligned}
$$

$$\mathit{Random}_{N:M} \overset{def}{=} \mathit{WebDyno}[N] \parallel \mathit{WorkerDyno}[M] \underset{\mathcal{L}_{random}}{\bowtie} (\mathit{WebRouter}_0 || \mathit{WorkerRouter}_0)$$

where $\mathcal{L}_{random} = \{\mathit{assign}_{web}, \mathit{assign}_{worker}, \mathit{migrate}\}$

Figure 6.4: PEPA model for random Heroku routing

6.2.2 Smart Routing Policy

The modelling of the dyno components is not substantially different from that of the random routing case, as both web and worker dynos are characterised by the same states and the same rates. The only difference is that we now have two distinct action types for assigning a job to a dyno. We want to capture the fact that a job may be either assigned to an idle dyno, or enqueued to an occupied dyno. Regarding the web dynos, only a *WebDyno* component will now be able to perform an $assign_{web}$ activity, as it denotes that the dyno is idle. For a $WebDyno_i$ component, which denotes an occupied web dyno with i requests in its local queue, jobs can only be enqueued as follows:

$$\begin{aligned}
WebDyno &\overset{def}{=} (assign_{web}, \top).WebDyno_0 \\
WebDyno_i &\overset{def}{=} (service, r_{web}).WebDyno_{ia} \\
&+ (migrate, r_{migrate}).WebDyno_{i-1} \\
&+ (enqueue_{web}, \top).WebDyno_{i+1}
\end{aligned}$$

Similarly, an $assign_{worker}$ activity can only be performed by *WorkerDyno*, while for the $WorkerDyno_i$ component we have:

$$\begin{aligned}
WorkerDyno &\overset{def}{=} (assign_{worker}, \top).WorkerDyno_0 \\
WorkerDyno_i &\overset{def}{=} (service, r_{worker}).WorkerDyno_{ia} \\
&+ (enqueue_{worker}, \top).WorkerDyno_{i+1}
\end{aligned}$$

The choice between assignment or placement in the local queue is a responsibility of the routing policy.

The smart routing policy consists of simply directing a request to a dyno that is available. If more than one dyno is available, then the router will randomly select a dyno. If there are no dynos available at a certain moment, then the request will be randomly enqueued to any dyno. The routing algorithm involves a deterministic step, which is the dyno availability check. Such a deterministic behaviour cannot be directly modelled according to the standard definition of PEPA. What we can do instead is to probabilistically favour assigning jobs to free dynos rather than placing then in queues. The idea is that the router will delay directing a request until a dyno is available. This delay should not be infinite however; if too many requests arrive, then the router will decrease its queue length by directing the requests to random dynos.

Regarding the *WebRouter* component, let n be the maximum queue length. Then for any queue length $i < n$, the requests will be assigned to web dynos that can perform an $assign_{web}$ activity; that means that the dyno in question is idle.

$$\begin{aligned}
WebRouter_0 &\stackrel{def}{=} (request, r_{request}).WebRouter_1 \\
WebRouter_i &\stackrel{def}{=} (request, r_{request}).WebRouter_{i+1} \\
&+ (assign_{web}, r_{assign}).WebRouter_{i-1}
\end{aligned}$$

If the queue length reaches its maximum size n, that probably means that no dyno has been available for a long time; it is then acceptable to send the request to the queue of any dyno. Let $WebRouter_n$ represent the state at which the corresponding router queue is full. Then $WebRouter_n$ will either assign or enqueue a request.

$$\begin{aligned}
WebRouter_n &\stackrel{def}{=} (request, r_{request}).WebRouter_n \\
&+ (assign_{web}, r_{assign} \times 0.5).WebRouter_{n-1} \\
&+ (enqueue_{web}, r_{assign} \times 0.5).WebRouter_{n-1}
\end{aligned}$$

By using similar reasoning, the migration queue on the router side will be modified as follows:

$$\begin{aligned}
WorkerRouter_0 &\stackrel{def}{=} (migrate, \top).WorkerRouter_1 \\
WorkerRouter_i &\stackrel{def}{=} (migrate, \top).WorkerRouter_{i+1} \\
&+ (assign_{worker}, r_{assign}).WorkerRouter_{i-1}
\end{aligned}$$

$$\begin{aligned}
WorkerRouter_n &\stackrel{def}{=} (migrate, \top).WorkerRouter_n \\
&+ (assign_{worker}, r_{assign} \times 0.5).WorkerRouter_{n-1} \\
&+ (enqueue_{worker}, r_{assign} \times 0.5).WorkerRouter_{n-1}
\end{aligned}$$

where n denotes the maximum queue length, and $i < n$.

To summarise, when the queue on the router part is not full, then the router works according to its "smart" mode of operation; it directs any requests to idle dynos only. Any new requests will have to wait in the router queue before being assigned. However, if the router queue reaches maximum capacity, this is an indication that the system is congested, suggesting that there are no idle dynos available. The router will then enter its "random" mode of operation, as it will decrease its queue by randomly directing requests to any dyno. That is captured by the fact that $enqueue_{web}$ and $enqueue_{worker}$ can only be performed if the corresponding router queue is full. The complete model for the smart routing policy is shown in Figure 6.5.

$$\begin{aligned}
WebDyno &\overset{def}{=} (assign_{web}, \top).WebDyno_0 \\
WebDyno_0 &\overset{def}{=} (service, r_{web}).WebDyno_{0a} + (migrate, r_{migrate}).WebDyno \\
&+ (enqueue_{web}, \top).WebDyno_1 \\
WebDyno_{0a} &\overset{def}{=} (response, r_{response}).WebDyno \\
WebDyno_1 &\overset{def}{=} (service, r_{web}).WebDyno_{1a} + (migrate, r_{migrate}).WebDyno_0 \\
WebDyno_{1a} &\overset{def}{=} (response, r_{response}).WebDyno_0
\end{aligned}$$

$$\begin{aligned}
WorkerDyno &\overset{def}{=} (assign_{worker}, \top).WorkerDyno_0 \\
WorkerDyno_0 &\overset{def}{=} (service, r_{worker}).WorkerDyno_{0a} + (enqueue_{worker}, \top).WorkerDyno_1 \\
WorkerDyno_{0a} &\overset{def}{=} (response, r_{response}).WorkerDyno \\
WorkerDyno_1 &\overset{def}{=} (service, r_{worker}).WorkerDyno_{1a} \\
WorkerDyno_{1a} &\overset{def}{=} (response, r_{response}).WorkerDyno_0
\end{aligned}$$

$$\begin{aligned}
WebRouter_0 &\overset{def}{=} (request, r_{request}).WebRouter_1 \\
WebRouter_1 &\overset{def}{=} (request, r_{request}).WebRouter_2 + (assign_{web}, r_{assign}).WebRouter_0 \\
WebRouter_2 &\overset{def}{=} (request, r_{request}).WebRouter_3 + (assign_{web}, r_{assign}).WebRouter_1 \\
WebRouter_3 &\overset{def}{=} (request, r_{request}).WebRouter_3 \\
&+ (assign_{web}, r_{assign} \times 0.5).WebRouter_2 \\
&+ (enqueue_{web}, r_{assign} \times 0.5).WebRouter_2
\end{aligned}$$

$$\begin{aligned}
WorkerRouter_0 &\overset{def}{=} (migrate, \top).WorkerRouter_1 \\
WorkerRouter_1 &\overset{def}{=} (migrate, \top).WorkerRouter_2 + (assign_{worker}, r_{assign}).WorkerRouter_0 \\
WorkerRouter_2 &\overset{def}{=} (migrate, \top).WorkerRouter_3 + (assign_{worker}, r_{assign}).WorkerRouter_1 \\
WorkerRouter_3 &\overset{def}{=} (migrate, \top).WorkerRouter_3 \\
&+ (assign_{worker}, r_{assign} \times 0.5).WorkerRouter_2 \\
&+ (enqueue_{worker}, r_{assign} \times 0.5).WorkerRouter_2
\end{aligned}$$

$$Smart_{N:M} \overset{def}{=} WebDyno[N] \parallel WorkerDyno[M] \underset{\mathcal{L}_{smart}}{\bowtie} (WebRouter_0 || WorkerRouter_0)$$

$$\mathcal{L}_{smart} = \{assign_{web}, enqueue_{web}, assign_{worker}, enqueue_{worker}, migrate\}$$

Figure 6.5: PEPA model for smart Heroku routing

6.3 Evaluation of Routing Policies

In this section, we present some experimental results in order to compare the two routing policies, and demonstrate how our approaches are affected as the scale of the model is increased. Medium sized models featuring a few millions of states can be successfully aggregated approximately in a compositional way. One limitation of approximate aggregation however, is that it is not as readily applicable if the scale of the model is just too large. In that case, reducing the state-space to a manageable size would mean that components are simply aggregated too much, therefore introducing erroneous behaviour in the model. However, simulation can still be an effective way to explore the stochastic properties of the system, and the trajectory sampling technique that we propose can deliver a reasonable speed-up in the analysis time.

6.3.1 Experimentation with the Workload

In this section, we experimentally evaluate how the routing policies considered respond to different workloads. We consider a medium-sized system, whose components are appropriate for compositional aggregation, and we evaluate the effect of aggregation approaches on the system behaviour.

More specifically, we consider a system featuring 8 web dynos and 8 worker dynos. We have two models that implement the two routing policies; these are $\mathit{Random}_{8:8}$ and $\mathit{Smart}_{8:8}$. By approximately reducing the components $\mathit{WebDyno}[8]$ and $\mathit{WorkerDyno}[8]$ to 60% of their original size each, the global state-space is effectively reduced to 36%. The original and the aggregated versions of the models have been solved for their transient and steady-state behaviour via the sparse engine of the PRISM model checker [58]. The steady-state distribution has been calculated using the Gauss-Seidel method. The transient probabilities have been calculated via the uniformisation method. The experiments have been performed in an Intel® Xeon$^{\mathrm{TM}}$ E5430 @ 2.66GHz PC running Ubuntu Linux.

The running times for $\mathit{Random}_{8:8}$ and $\mathit{Smart}_{8:8}$ are summarised in Tables 6.2 and 6.3 correspondingly. According to these tables, the state-space reduction resulted in the expected reduction in the analysis time, for both approximate aggregation methods, based on NCD and quasi-lumpability correspondingly. The most interesting thing regarding the running times is that the time needed to approximate the state-space is trivial compared to the time saved.

The next thing to see is whether the approximate results obtained provide us with

Table 6.2: Running Times for $Random_{8:8}$

	Original	Quasi-Lumpability (Compositional)	NCD (Compositional)
Approximation	-	3 sec	3 sec
PRISM Loading	10000 sec	4000 sec	5000 sec
Transient Solution[a]	29000 sec	13000 sec	16000 sec
Steady-State solution	250 sec	200 sec	200 sec
Total Time	39250 sec	17200 sec	21200 sec
Number of states	3920400	1411344	1411344

[a] 50 points: $0 \leq t \leq 4$

Table 6.3: Running Times for $Smart_{8:8}$

	Original	Quasi-Lumpability (Compositional)	NCD (Compositional)
Approximation	-	3 sec	3 sec
PRISM Loading	11000 sec	5000 sec	5000 sec
Transient Solution[b]	32000 sec	14000 sec	15000 sec
Steady-State solution	370 sec	230 sec	300 sec
Total Time	43370 sec	19230 sec	20300 sec
Number of states	3849444	1401852	1394760

[b] 50 points: $0 \leq t \leq 4$

reliable information regarding the properties of the routing policies. In terms of the current section, we have experimented with two different values for the request rate 40 and 60, in order to observe how the two routing policies respond to different workloads. The effects of each policy should be reflected in the average dyno queue length and in the number of dynos that remain idle. As a general remark on the results that follow, the quasi-lumpability approach provides much more accurate results compared to the NCD-based method.

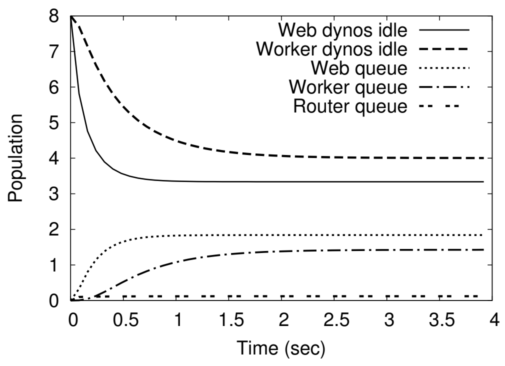
Figure 6.6 outlines the transient behaviour for request arrival rate equal to 40 sec^{-1}, or 2400 requests per minute. The data plotted depicts how the average population of idle dynos and average local queue lengths change during the first four seconds of the system being online. We can see that after these four seconds, the system appears to be in steady state. The left column of plots presents results for the random routing policy, while the plots on the right column correspond to smart routing.

Judging by the first two plots in Figures 6.6(a) and 6.6(b), which show the results of the original model, part of the system is underused for both smart and random routing, as we have a significant number of idle dynos in both cases. However, the average dyno queue lengths are considerably higher for random routing. This means that some requests might have to wait in the queue, while there are dynos available. That is not the case for smart routing however, where the dyno queues are almost empty. In other words, the smart routing fully exploits the capacity of the Heroku configuration, in contrast with the random routing policy.

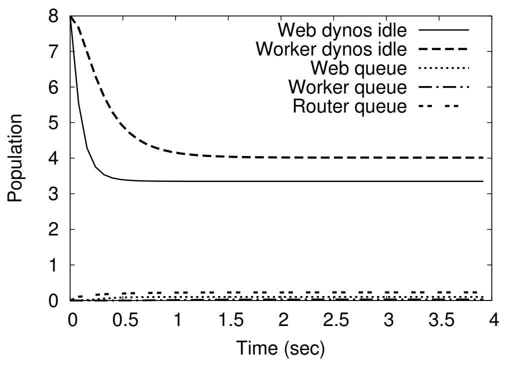
Regarding the results of the quasi-lumpability method in Figures 6.6(c) and 6.6(d), they seem to qualitatively agree with the true results, although the numerical values for the average idle dynos do not exactly match. Nevertheless, the qualitative difference between random and smart routing with respect to the average dyno queue length is adequately captured. We think that the modest reduction in accuracy was a worthwhile price to pay, especially given the substantial reduction in analysis time (Tables 6.2 and 6.3). The same can be said to a lesser extent for the NCD-based approach in Figures 6.6(e) and 6.6(f), where a greater amount of error is introduced in this example. However, the qualitative difference with respect to the queue lengths is still evident.

In the experiment summarised in Figure 6.7, we investigate how the routing policies are affected by a higher workload, by increasing the request arrival rate to 60 sec^{-1}, or 3600 requests per minute. According to Figures 6.7(a) and 6.7(b), which depict the behaviour of the unreduced models, the system usage is similar for both random and smart routing, as can see by the numbers of idle web and worker dynos. For the smart system, the dyno queues have significantly shorter length when compared to the random routing policy, implying that the requests wait less time until they are serviced. As a final comment, we can say that a request arrival rate to 40 is probably the most that the smart routing policy can effectively handle for the number of dynos considered.

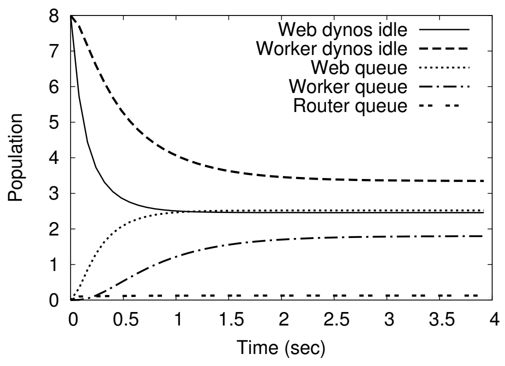
The results of the quasi-lumpability approach in Figures 6.7(c) and 6.7(d) give a similar picture. The relation between idle dynos and the corresponding average queue lengths has been portrayed accurately enough to show the different behaviour of the two routing policies. Regarding the NCD-based approach however in Figures 6.7(e) and 6.7(f), the results are less accurate as the queue lengths are apparently underestimated for both routing policies. Moreover, it seems that the approximated systems via the NCD approach do not capture the transient dynamics of the system at all. We think that this is a clear indication that this particular system should not be characterised as nearly-completely decomposable.



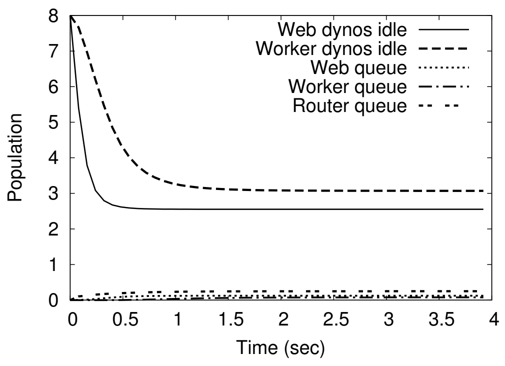
(a) Random routing (Original)



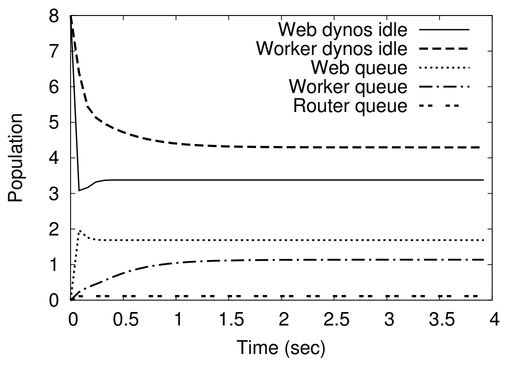
(b) Smart routing (Original)



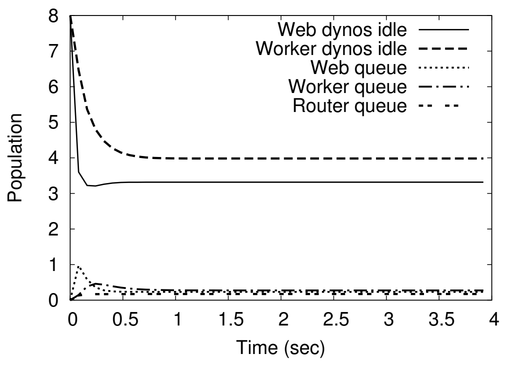
(c) Random routing (Quasi-lumpability)



(d) Smart routing (Quasi-lumpability)



(e) Random routing (NCD)

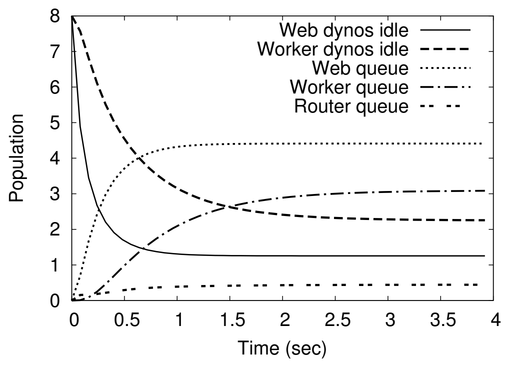


(f) Smart routing (NCD)

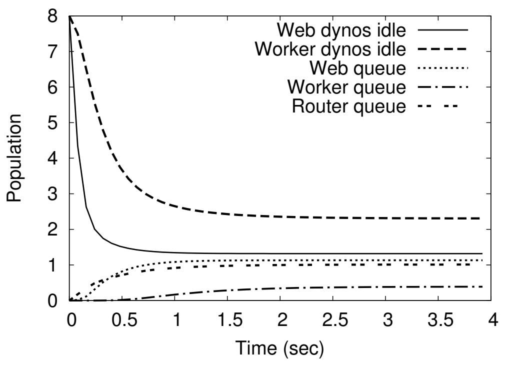
Figure 6.6: $Random_{8:8}$ and $Smart_{8:8}$ results for $r_{request} = 40$

To summarise, the smart routing policy results in better utilisation of the system resources compared to random routing, judging by the number of requests that remain in the queues at the dyno level. Smart routing results in a significantly shorter average queue length, regardless of the workload.

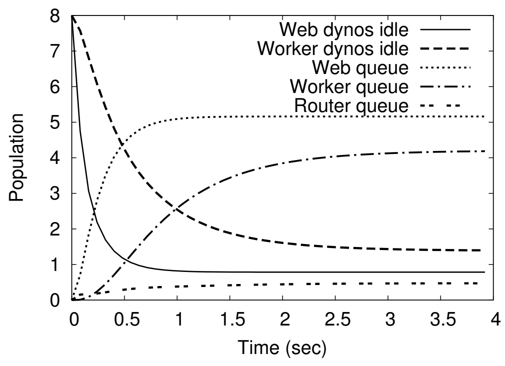
Applying compositional aggregation, and the quasi-lumpability approach in partic-



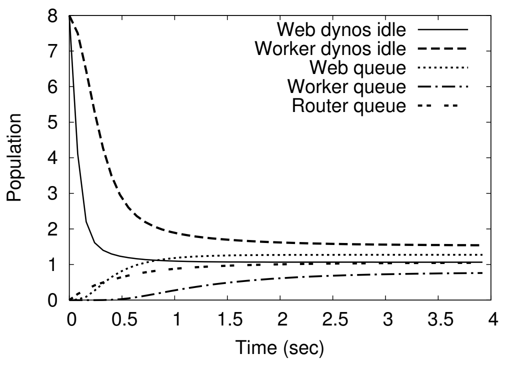
(a) Random routing (Original)



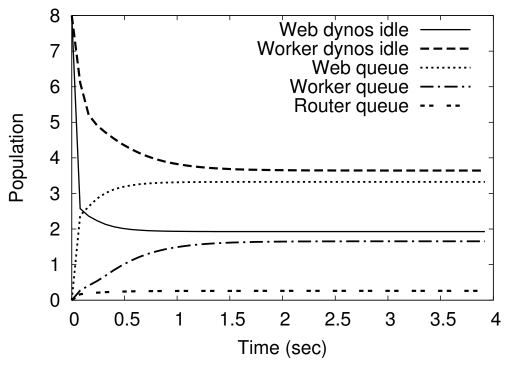
(b) Smart routing (Original)



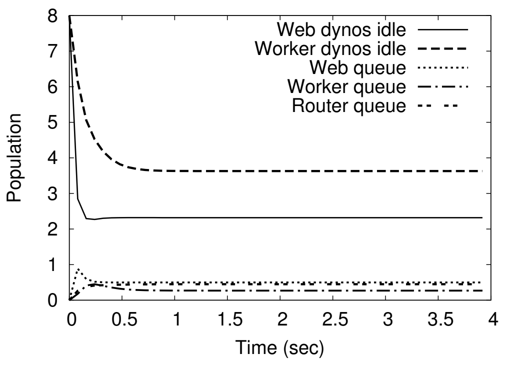
(c) Random routing (Quasi-lumpability)



(d) Smart routing (Quasi-lumpability)



(e) Random routing (NCD)



(f) Smart routing (NCD)

Figure 6.7: $Random_{8:8}$ and $Smart_{8:8}$ results for $r_{request} = 60$

ular, has led us to the same conclusion at a significantly lower cost. However, the NCD-based approach has been considerably less accurate compared to the quasi-lumpability approach for the Heroku example. In fact, both approaches rely on assumptions that may or may not hold for a specific model. One assumption is that there is a partition of the state-space with respect to which the model is either quasi-lumpable or nearly completely decomposable. Our position is that quasi-lumpability should be able to capture a wider range of approximate equivalences, simply because any nearly completely decomposable model is essentially quasi-lumpable as well. That does not mean however that the quasi-lumpability approach as defined in this thesis should always give better results, as it is sub-optimal. We have extensively discussed that the partition provided is only an approximation to what could be an optimum partition from a quasi-lumpability point of view.

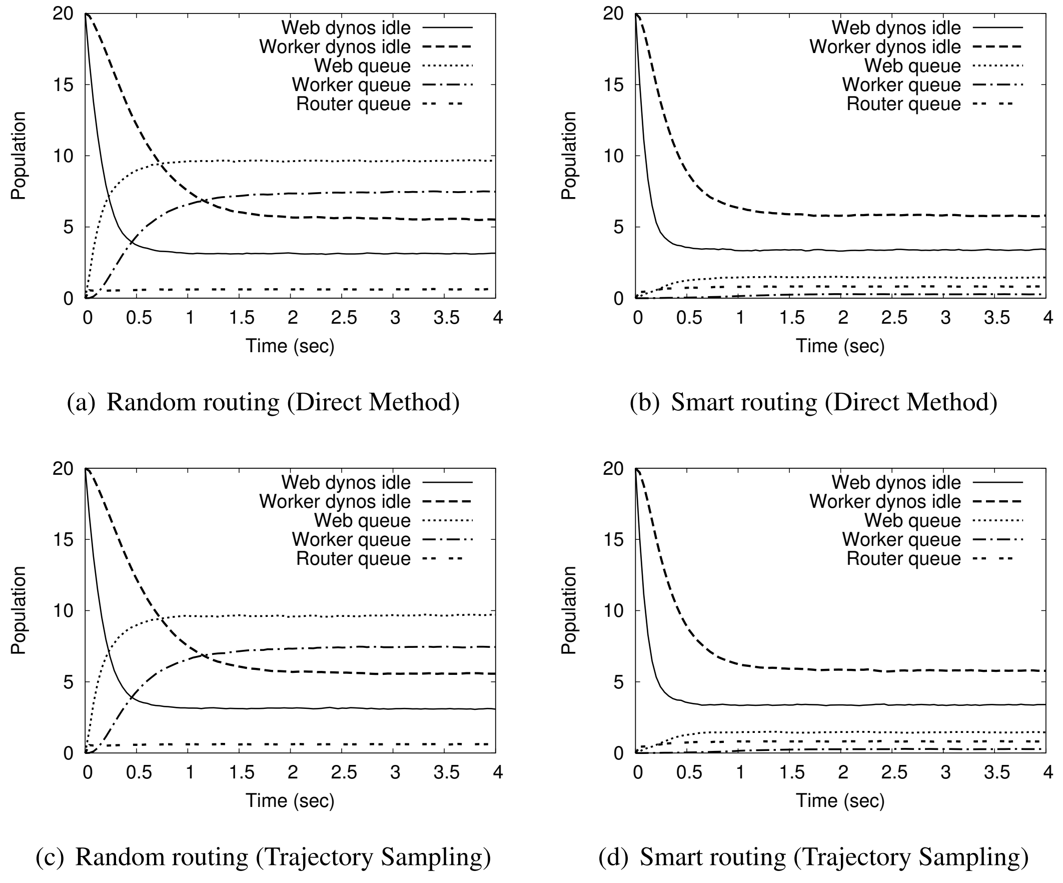
As a final remark, there is a question that has not been still answered yet. That is how many dynos are needed to service 9000 requests per minute. Scaling the model further will result in excessive explosion of the state-space, therefore further reduction is required to keep the state-space manageable. The applicability of approximate aggregation is limited by the properties of the model. Although the quasi-lumpability approach captured the qualities of the two routing policies in question, the numerical diversities compared to the original model have not been insignificant. Therefore, it is uncertain how the quasi-lumpability approach would respond to a greater reduction of the state-space. Nevertheless, increasing the scale of the system offers a great opportunity to apply a stochastic simulation approach, trajectory sampling in particular, which is discussed in the next section.

6.3.2 Experimentation with the System Size

The medium-sized system that we have examined in the previous section has shown that there is a significant difference in terms of performance between the two routing policies considered. Our objective now is to investigate how many dynos are required to service 9000 requests per minute, translated into a request arrival rate equal to 150 sec^{-1}, which is the reported workload for Rap Genius.

In this experiment, we consider a fixed arrival rate equal to 150, while we perform experimentation with the size of the system, so as to determine how many dynos have to be leased, so that both the number of idle dynos and the queue length in the dyno level are minimised. We have to scale the service to a larger number of dynos than we have considered so far. We use the models of Figures 6.4 and 6.5 to study the random and the smart routing policies correspondingly. This time however, rather than solving for the transient probabilities after applying compositional aggregation, we shall simply simulate the system. We shall apply trajectory sampling simulation (TSS), with parameter $p = 0.1$ for the geometric approximation, which has been the value of choice in Section 5.3.

Figure 6.8 outlines the transient behaviour for 20 web dynos and 20 worker dynos. Each sub-figure describes how the average population of idle dynos and the average queue lengths at the dyno-level change through time. More specifically, in Figure 6.8(d) we see that we have only a small number of idle dynos, while the number of jobs queued at the dyno-level remains small. Therefore, the system of this size has been found to be adequate to service 9000 requests per minute by using the smart routing policy. According to Figure 6.8(c) however, the queue lengths are considerably larger for the random policy. Simply, more dynos are needed to decrease the number of requests waiting in the dyno queues.

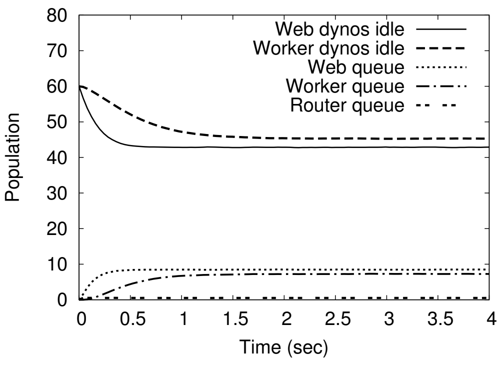


(a) Random routing (Direct Method)
(b) Smart routing (Direct Method)
(c) Random routing (Trajectory Sampling)
(d) Smart routing (Trajectory Sampling)

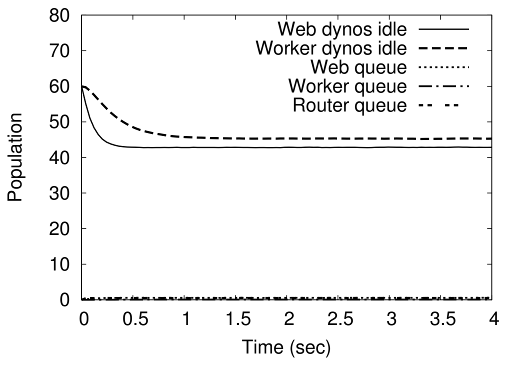
Figure 6.8: $Random_{20:20}$ and $Smart_{20:20}$ results for $r_{request} = 150$ (10^5 simulation runs)

We have also considered a system with 60 web dynos and 60 worker dynos, whose results are summarised in Figure 6.9. For the random routing policy in Figure 6.9(c), we have relatively small but non-zero number of requests in the dyno queues. It appears that a random routing policy has a negative impact on the request waiting time, regardless of the size of the system. The picture is quite different for the smart policy in Figure 6.9(d), where the almost no request is waiting. In both cases though, a large part of the system remains idle, meaning that the use of 60 dynos of each kind is simply a waste of resources considering the given workload.

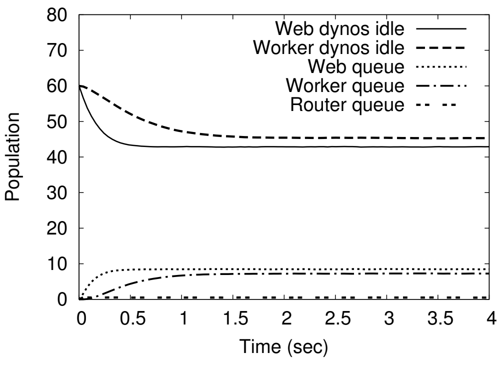
Apparently, a system of 20 web and 20 worker dynos featuring a smart routing policy should be enough to service the typical workload of a website such as Rap Genius. Replacing smart with a random policy will only increase the number of dynos required to service the same workload at the same rate, and therefore diminish the quality of service provided to the clients.



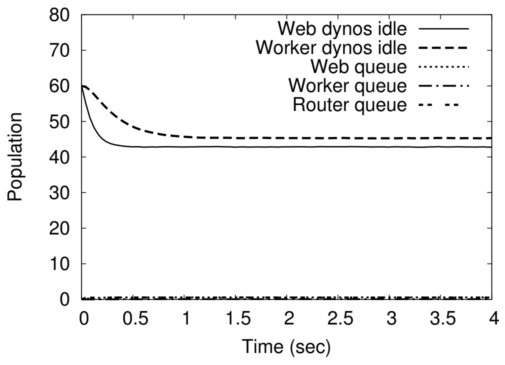
(a) Random routing (Direct Method)



(b) Smart routing (Direct Method)



(c) Random routing (Trajectory Sampling)



(d) Smart routing (Trajectory Sampling)

Figure 6.9: $Random_{60:60}$ and $Smart_{60:60}$ results for $r_{request} = 150$ (10^5 simulation runs)

Regarding the approximation quality of our trajectory sampling algorithm, its results are practically identical to the output of the direct method. In fact, this outcome was anticipated, as it is compliant with the conclusions of Chapter 5. We have characterised trajectory sampling simulation as an almost exact method, in the sense that it can be arbitrarily precise. We have seen experimentally observed in Section 5.3 that

Table 6.4: Running times in seconds for 10^5 simulation runs

	ODM	Trajectory Sampling
$Random_{20:20}$	360	293
$Random_{60:60}$	360	300
$Smart_{20:20}$	310	268
$Smart_{60:60}$	315	250

TSS with parameter $p = 0.1$ for the geometric approximation produces very accurate results, a fact that is also reflected in the current section.

Finally, Table 6.4 compares the running times for 10^5 simulation runs between the optimised direct method (ODM) and TSS. The experiments have been performed in an Intel® Xeon™ E5410 @ 2.33GHz PC running Scientific Linux 6. For all of the models considered, TSS results in an improvement around $15 \sim 20\%$ with respect the total running time, in agreement with the conclusions of Chapter 5.

6.4 Summary

In this chapter, we have demonstrated how the approximation methodologies discussed in this thesis can be useful tools to investigate problems of the real world. The problem under consideration has been to evaluate routing policies for the Heroku PaaS provider. The example used has been motivated by a particular incident involving the Rap Genius website, where a change in the routing policy has been reported to negatively affect the quality of service. It has to be emphasised that any conclusions regarding the routing policies for Heroku and Rap Genius are liable to any assumptions made, including the exponentially distributed events and the rates used. Although our model does not aspire to be an accurate representation of Rap Genius, we think that it is realistic representation of a system of that scale.

It has been observed that a smart routing policy results in a significantly smaller number of requests waiting to be serviced, compared to a random policy. Regarding approximation quality, the results obtained by the approaches proposed in this thesis, namely compositional approximate aggregation and trajectory sampling simulation, have been accurate enough to support valid conclusions in most cases.

On compositional aggregation, we have to comment that approximation quality for the Heroku example has not been the same for the two aggregation approaches considered. More specifically, the NCD-based approach has found to be significantly less accurate than the quasi-lumpability approach. This is a clue that quasi-lumpability should be able to describe a wider family of approximate state equivalences that can be exploited in terms of aggregation. However, it is not suggested that the quasi-lumpability approach should be more or less accurate than the NCD-based method in the general case. The suitability of either method depends on the properties of the given model, and whether the underlying CTMC is close to being either quasi-lumpable or nearly completely decomposable. This is in fact an inherent limitation of any approximate state-space aggregation method, for which the existence of an appropriate partition has been a key assumption. Any attempt to approximate a state-space for which there not a partition good enough is susceptible to errors.

Increasing the scale of the system too much has made the state-space unmanageable even for a compositional aggregation approach. We have resorted to simulation to explore the stochastic properties of very large systems, since no explicit state-space representation is required. Our trajectory sampling approach (TSS) has found to be remarkably accurate compared to exact simulation. This can be attributed to the fact that TSS produces very detailed simulation trajectories, which involve all of the events that take place during the life-time of the stochastic process. The efficiency improvement has found to be less impressive, which is essentially a limitation of any method that aspires to be exact. Nevertheless, we think that even the smallest efficiency gain is important, especially when the price to pay in terms of accuracy is virtually zero, as in the case of TSS.

Chapter 7

Conclusions

Markov chains have been used for many years for exploring the dynamic properties of systems that exhibit stochastic behaviour. Modelling formalisms that generate Markov chains given a high-level specification exist in abundance, including Petri nets, queueing networks or stochastic automata networks. In this thesis, we have focused on stochastic process algebras, PEPA and Bio-PEPA in particular, which offer a compositional framework to Markovian modelling.

Compositionality is the most important quality of stochastic process algebras compared to other modelling paradigms. The ability to describe systems as collections of interacting components provides an effective way to describe complex systems using a minimal specification. For that reason, Markovian process algebras have proven to be valuable tools for performance modelling. They are however prone to the problem of state-space explosion, meaning that even apparently simple models may generate very large Markov chains.

The main contribution of this thesis is to investigate approximation methods that preserve the stochastic properties of Markovian process algebra models. We have discussed two main subjects: compositional state-space aggregation and stochastic simulation via trajectory sampling.

7.1 Contributions on Approximate Aggregation

The problem of Markov chain aggregation has been defined as a problem of partitioning the state-space in such a way that similar states are grouped together. The fundamental assumption is that similar states could be sufficiently represented as a whole by an aggregated state. The set of aggregated states will form the state-space of a reduced model, whose behaviour is assumed to approximate the original Markov chain. To summarise our contributions, we have investigated approximate aggregation methodologies for Markov chains, and we have proposed compositional state-space aggregation as a means of efficient model reduction.

7.1.1 Approximate Aggregation Methodologies

The notion of NCD has been traditionally used to describe state similarity in Markov chains. A Markov chain is said to consist of nearly-completely decomposable sets of states, or classes, if there are strong interactions within those classes, and weak interactions among the classes. It is hoped that the interactions within the classes can be abstracted away, resulting in a reduced model which will be approximate if it has to be a Markov chain. The identification of nearly-completely decomposable sets can be achieved by exploiting the spectral properties of transition probability matrices. We have shown that the notion of NCD is strongly related to the principles that spectral clustering relies on. Thus, we have been able to adopt some well-known results and methodologies from the field of spectral clustering, in order to identify nearly-completely decomposable partitions for Markov chains.

We have argued that there should be a more appropriate measure than NCD to define approximate state similarity in a Markov chain. The concept of lumpability has been the starting point of our discussion, as it captures state equivalence in a way similar to probabilistic bisimulation. The idea is that equivalent states should have similar behaviour with respect to a partition of the state-space, in contrast to being tightly coupled as NCD implies. An approximate version of lumpability, namely quasi-lumpability, has also been known in the literature. Our contribution consists of a strategy to discover state-space partitions that are close to being quasi-lumpable. We have defined a measure to express the degree of state similarity that is compatible with the notion of quasi-lumpability. The biggest challenge for our measure has been that it is not constant for each pair of states, as it depends on the partition. We have shown that an appropriate adaptation of a clustering algorithm will essentially minimise an upper bound of this quasi-lumpability measure for states that are assigned to same partition. The use of such an upper bound has made the problem manageable, but it has also rendered our approach sub-optimal, meaning that the most appropriate partition may not always be discovered.

Provided that an appropriate state-space partition has been obtained by any of the two aforementioned partitioning approaches, the final step is to summarise the transition probabilities from each class to another. The objective is to construct a Markov chain of reduced size that approximates the original model. Our suggestion is that the class-to-class transition probabilities should be calculated as the average transition probabilities with respect to the states of each class. We have proven that such an approximately aggregated Markov chain will be within the bounds obtained by stochastic comparison of Markov chains.

Regarding the experimental results, we could not conclude which of the two partitioning approaches is superior. We have observed that different models favour one or the other approach. Regarding the upper and lower bounds obtained by stochastic comparison, we have seen that a good partition, that is one that results in a low approximation error, does not necessarily result in tight bounds. The tightness of the bounds is susceptible to the bounding algorithm.

7.1.2 Compositional Aggregation

Compositional aggregation as presented is strongly connected with the process algebra modelling paradigm assumed. We have treated PEPA components as labelled CTMCs, and we have identified an approximate notion of state equivalence for components that is desirable for state-space aggregation. We have called this "modified quasi-strong equivalence", which is assumed to be approximated by quasi-lumpability, if the component in question has a large amount of individual activities.

In order to describe systems that consist of approximately aggregated components, we have proposed two alternative approaches. The first approach is an adaptation of the Kronecker representation proposed by Hillston & Kloul [50]. We have also discussed an issue of such an approach based on Kronecker algebra, which is the inclusion of unreachable states in the final generator matrix. Even if those states do not affect the behaviour of the constructed model, they do have an effect on the size of the state-space produced. To overcome this issue, we have also produced explicit structured operational semantics for approximately aggregated components. In this way, we could construct an aggregated version of the derivation graph for a PEPA model, and therefore avoid unreachable states in the generator matrix.

By applying approximate aggregation compositionally, it was possible to produce reasonable approximations for a class of multi-scale models. The approximation quality for the compositional aggregation has not been inferior to globally applied aggregation. We do not expect that this finding generalises for arbitrary models however; the models assumed feature components that have a significant amount of individual actions. Therefore, there has been enough data for the partitioning approaches to work with. The most important thing however is that the computational overhead of compositional approximation has been only a fraction of the total analysis time. In contrast, the overhead of globally applied aggregation renders such an approach unreasonable, due to the inherently high complexity of the partitioning strategies discussed.

Despite the potential of compositional aggregation, we have acknowledged that it is associated with certain limitations. First of all, the components to be approximated are assumed to exhibit a sufficient amount of individual behaviour. Secondly, models featuring unbounded state-spaces, which is typical case for Bio-PEPA models for example, are unlikely to be approximated by a model reduction approach.

7.2 Contributions on Simulation

Simulation is always a relevant approach to study the stochastic properties of systems, regardless of their size, since an explicit state-space representation is not required. We have explored the possibility of accelerating the stochastic simulation process by reducing the amount of random numbers generated. The trajectory sampling simulation approach that we have proposed, as its name implies, samples from the trajectory rather than the transition space. We have shown that it is possible to use a single random number to determine an entire sequence of transitions. Regarding the exponential delays associated with the CTMC transitions, these have been approximated by a sequence of geometric random variables. This time discretisation allowed us to employ the trajectory sampling technique to determine the duration for a sequence of transitions using a single random sample.

We have characterised our simulation method as almost exact, since it has been shown to be arbitrarily precise. The only source of approximation is the use of geometric distributions to determine time; we have shown that approximation quality is controlled by the time-discretisation parameter, which is determined by the user. The most important quality of our method is that its performance and accuracy are independent of the model. In contrast, approximate simulation approaches skip some of the simulation events by exploiting certain model properties. Such methods are inherently more efficient, however TSS can be thought of as an alternative to those in cases where the assumptions that they rely on are incompatible with the model in question. The experimental results have shown TSS to be consistently efficient and accurate for all of the models tested.

7.3 Future Work

In this section, we outline directions of future research, which can potentially establish compositional aggregation as a generic framework for efficient model reduction.

Tool Support

We have assumed that decisions regarding component aggregation are made by the modeller, a fact that essentially renders compositional aggregation a human-driven process. Such decisions involve the selection of the components to be approximated, and the number of classes to which each component will be reduced. Tool support is necessary for the modeller to rationalise any choices concerning aggregation.

For example, the modeller can be guided by a user interface through the different possibilities of aggregation and the effect that these will have in the modelling. Given a particular setting, the user should be presented with information regarding the expected state-space reduction and the expected time required to approximate the corresponding components. The modeller would then be able to judge whether aggregating a collection of components is a practical strategy to look into the behaviour of a system.

As with any approximation methodology, approximate aggregation is prone to errors, unless the system under consideration has a very strong behavioural pattern that can be discovered. The current state of our work does not involve any strategy to assess the approximation quality of a partition *a priori* i.e. before solving the model. Such a partition characterisation is a great challenge, however we think that it can create many possibilities that will extend the impact of this research. For example, the ability to evaluate partitions can be used as a guide by the modeller to determine the appropriate number of classes for a component, by means of experimentation. Another possibility is to produce error expectations, a fact that will significantly increase the confidence in the results given by aggregated models.

Improving Partitioning of PEPA Components

One limitation of the current approach for compositional aggregation is that PEPA components are partitioned by taking into account their individual activities only, while their shared behaviour is ignored. This convention imposed a requirement for the components to be approximated: it is assumed that their behaviour is dominated by their individual activities. The issue with the shared activities is that their rates depend on the global state of the system. If we could formulate expectations for those rates, then it would be possible to include shared activities in the partitioning process as well. This could improve approximation quality, but most importantly, it would render our method applicable to a larger family of PEPA models.

Improving Quasi-Lumpability Aggregation

We think that approximation quality can be further improved by the refinement of our partitioning approach that relies on the concept of quasi-lumpability. It is our position that quasi-lumpability is more appropriate as a measure of behavioural similarity between states in a Markov chain. The fact that we could define a sub-optimal quasi-lumpability aggregation approach which performs just as well as the NCD-based approach, if not better in some cases, is a strong indication that quasi-lumpability is an appropriate criterion for Markov chain aggregation.

Appendix A

Bio-PEPA Models of Chapter 5

A.1 The Schlögl Model

Kinetic Parameters:

$$\begin{aligned} c_1 &= 3 \times 10^{-7}/2 \\ c_2 &= 10^{-4}/6 \\ c_3 &= 10^{-3} \\ c_4 &= 3.5 \end{aligned}$$

Initial Populations:

$$\begin{aligned} x_0 &= 250 \\ n_1 &= 100000 \\ n_2 &= 200000 \end{aligned}$$

Functional Rates:

$$\begin{aligned} r_1 &: c_1 \times B_1 \times X \times (X-1) \\ r_2 &: c_2 \times X \times (X-1) \times (X-2) \\ r_3 &: c_3 \times B_2 \\ r_4 &: c_4 \times X \end{aligned}$$

Species Components:

$$\begin{aligned}
X &\overset{def}{=} r_1\uparrow + r_2\downarrow + r_3\uparrow + r_4\downarrow \\
B_1 &\overset{def}{=} r_1\downarrow + r_2\uparrow \\
B_2 &\overset{def}{=} r_3\downarrow + r_4\uparrow
\end{aligned}$$

Model Component:

$$X[x_0] \underset{\{*\}}{\bowtie} B_1[n_1] \underset{\{*\}}{\bowtie} B_2[n_2]$$

A.2 The LacY Model

Initial Populations:

$$\begin{aligned}
PLac_0 &= 1 \\
RNAP_0 &= 3 \\
PLacRNAP_0 &= 0 \\
TrLacZ1_0 &= 0 \\
RbsLacZ_0 &= 0 \\
TrLacZ2_0 &= 0 \\
TrLacY1_0 &= 0 \\
RbsLacY_0 &= 0 \\
TrLacY2_0 &= 0 \\
Ribosome_0 &= 30 \\
RbsRibosomeLacZ_0 &= 0 \\
RbsRibosomeLacY_0 &= 0 \\
TrRbsLacZ_0 &= 0 \\
TrRbsLacY_0 &= 0 \\
LacZ_0 &= 0 \\
LacY_0 &= 0 \\
dgrLacZ_0 &= 0 \\
dgrLacY_0 &= 0 \\
dgrRbsLacZ_0 &= 0
\end{aligned}$$

$$
\begin{aligned}
dgrRbsLacY_0 &= 0 \\
lactose_0 &= 0 \\
product_0 &= 0
\end{aligned}
$$

Functional Rates:

$$
\begin{aligned}
r_1 &: 0.17 \times (PLac \times RNAP) \\
r_2 &: 10 \times PLacRNAP \\
r_3 &: 1 \times PLacRNAP \\
r_4 &: 1 \times TrLacZ1 \\
r_5 &: 0.015 \times TrLacZ2 \\
r_6 &: 1 \times TrLacY1 \\
r_7 &: 0.36 \times TrLacY2 \\
r_8 &: 0.17 \times (Ribosome \times RbsLacZ) \\
r_9 &: 0.17 \times (Ribosome \times RbsLacY) \\
r_{10} &: 0.45 \times RbsRibosomeLacZ \\
r_{11} &: 0.45 \times RbsRibosomeLacY \\
r_{12} &: 0.4 \times RbsRibosomeLacZ \\
r_{13} &: 0.4 \times RbsRibosomeLacY \\
r_{14} &: 0.015 \times TrRbsLacZ \\
r_{15} &: 0.036 \times TrRbsLacY \\
r_{16} &: 6.42 \times 10^{-5} \times LacZ \\
r_{17} &: 6.42 \times 10^{-5} \times LacY \\
r_{18} &: 0.3 \times RbsLacZ \\
r_{19} &: 0.3 \times RbsLacY \\
r_{20} &: 0.0005731 \times (LacZ \times lactose) \\
r_{21} &: 14 \times LacY
\end{aligned}
$$

Species Components:

$$
\begin{aligned}
PLac &\overset{def}{=} r_1 \downarrow + r_2 \uparrow + r_4 \uparrow \\
RNAP &\overset{def}{=} r_1 \downarrow + r_2 \uparrow + r_7 \uparrow \\
PLacRNAP &\overset{def}{=} r_1 \uparrow + r_2 \downarrow + r_3 \downarrow \\
TrLacZ1 &\overset{def}{=} r_3 \uparrow + r_4 \downarrow
\end{aligned}
$$

$$
\begin{aligned}
RbsLacZ &\overset{def}{=} r_4\uparrow \;+\; r_8\downarrow \;+\; r_{10}\uparrow \;+\; r_{12}\uparrow \;+\; r_{18}\downarrow\\
TrLacZ2 &\overset{def}{=} r_4\uparrow \;+\; r_5\downarrow\\
TrLacY1 &\overset{def}{=} r_5\uparrow \;+\; r_6\downarrow\\
RbsLacY &\overset{def}{=} r_6\uparrow \;+\; r_9\downarrow \;+\; r_{11}\uparrow \;+\; r_{13}\uparrow \;+\; r_{19}\downarrow\\
TrLacY2 &\overset{def}{=} r_6\uparrow \;+\; r_7\downarrow\\
Ribosome &\overset{def}{=} r_8\downarrow \;+\; r_9\downarrow \;+\; r_{10}\uparrow \;+\; r_{11}\uparrow\\
RbsRibosomeLacZ &\overset{def}{=} r_8\uparrow \;+\; r_{10}\downarrow \;+\; r_{12}\downarrow\\
RbsRibosomeLacY &\overset{def}{=} r_9\uparrow \;+\; r_{11}\downarrow \;+\; r_{13}\downarrow\\
TrRbsLacZ &\overset{def}{=} r_{12}\uparrow \;+\; r_{14}\downarrow\\
TrRbsLacY &\overset{def}{=} r_{13}\uparrow \;+\; r_{15}\downarrow\\
LacZ &\overset{def}{=} r_{14}\uparrow \;+\; r_{16}\downarrow \;+\; r_{20}\odot\\
LacY &\overset{def}{=} r_{15}\uparrow \;+\; r_{17}\downarrow \;+\; r_{21}\odot\\
dgrLacZ &\overset{def}{=} r_{16}\uparrow\\
dgrLacY &\overset{def}{=} r_{17}\uparrow\\
dgrRbsLacZ &\overset{def}{=} r_{18}\uparrow\\
dgrRbsLacY &\overset{def}{=} r_{19}\uparrow\\
lactose &\overset{def}{=} r_{20}\downarrow \;+\; r_{21}\uparrow\\
product &\overset{def}{=} r_{20}\uparrow
\end{aligned}
$$

Model Component:

$$
\begin{aligned}
&PLac[PLac_0] \underset{\{*\}}{\bowtie} RNAP[RNAP_0] \underset{\{*\}}{\bowtie} PLacRNAP[PLacRNAP_0]\\
&\quad \underset{\{*\}}{\bowtie} TrLacZ1[TrLacZ1_0] \underset{\{*\}}{\bowtie} RbsLacZ[RbsLacZ_0] \underset{\{*\}}{\bowtie} TrLacZ2[TrLacZ2_0]\\
&\quad \underset{\{*\}}{\bowtie} TrLacY1[TrLacY1_0] \underset{\{*\}}{\bowtie} RbsLacY[RbsLacY_0] \underset{\{*\}}{\bowtie} TrLacY2[TrLacY2_0]\\
&\quad \underset{\{*\}}{\bowtie} Ribosome[Ribosome_0] \underset{\{*\}}{\bowtie} RbsRibosomeLacZ[RbsRibosomeLacZ_0]\\
&\quad \underset{\{*\}}{\bowtie} RbsRibosomeLacY[RbsRibosomeLacY_0] \underset{\{*\}}{\bowtie} TrRbsLacZ[TrRbsLacZ_0]\\
&\quad \underset{\{*\}}{\bowtie} TrRbsLacY[TrRbsLacY_0] \underset{\{*\}}{\bowtie} LacZ[LacZ_0] \underset{\{*\}}{\bowtie} LacY[LacY_0]\\
&\quad \underset{\{*\}}{\bowtie} dgrLacZ[dgrLacZ_0] \underset{\{*\}}{\bowtie} dgrLacY[dgrLacY_0]\\
&\quad \underset{\{*\}}{\bowtie} dgrRbsLacZ[dgrRbsLacZ_0] \underset{\{*\}}{\bowtie} dgrRbsLacY[dgrRbsLacY_0]\\
&\quad \underset{\{*\}}{\bowtie} lactose[lactose_0] \underset{\{*\}}{\bowtie} product[product_0]
\end{aligned}
$$

A.3 The Goldbeter's Model

Kinetic Parameters:

$$\begin{aligned}
\Omega &= 6022 \\
vi &= 0.025 \times \Omega \\
kd &= 0.01 \\
kc &= 0.5 \times \Omega \\
v_1 &= 12 \times \Omega \\
v_2 &= 1.5 \\
v_3 &= 12 \\
v_4 &= 2 \\
k_1 &= 0.02 \times \Omega \\
k_2 &= 0.02 \times \Omega \\
k_3 &= 0.02 \times \Omega \\
k_4 &= 0.02 \times \Omega \\
vd_2 &= 0.0625 \\
kd_2 &= 0.02 \times \Omega
\end{aligned}$$

Functional Rates:

$$\begin{aligned}
a_1 &: vi \\
a_2 &: kd \times C \\
a_3 &: \frac{v_1 \times C}{kc + C} \times \frac{inactive\_M}{k_1 + inactive\_M} \\
a_4 &: \frac{v_2 \times active\_M \times enzyme}{k_2 + active\_M} \\
a_5 &: \frac{v_3 \times inactive\_X \times active\_M}{k_3 + inactive\_X} \\
a_6 &: \frac{v_4 \times active\_X \times enzyme}{k_4 + active\_X} \\
a_7 &: \frac{C \times vd_2 \times active\_X}{C + kd_2}
\end{aligned}$$

Species Components:

$$
\begin{aligned}
C &\overset{def}{=} a_1 \uparrow C \quad + \quad a_2 \downarrow C \quad + \quad a_7 \downarrow C \quad + \quad a_3 \oplus C \\
inactive\_M &\overset{def}{=} a_4 \uparrow inactive\_M \quad + \quad a_3 \downarrow inactive\_M \\
active\_M &\overset{def}{=} a_3 \uparrow active\_M \quad + \quad a_4 \downarrow active\_M \quad + \quad a_5 \oplus active\_M \\
inactive\_X &\overset{def}{=} a_6 \uparrow inactive\_X \quad + \quad a_5 \downarrow inactive\_X \\
active\_X &\overset{def}{=} a_5 \uparrow active\_X \quad + \quad a_6 \downarrow active\_X \quad + \quad a_7 \oplus active\_X \\
enzyme &\overset{def}{=} a_4 \oplus enzyme \quad + \quad a_6 \oplus enzyme
\end{aligned}
$$

Model Component:

$$
\begin{aligned}
C[60] &\underset{\{*\}}{\bowtie} active\_M[60] \underset{\{*\}}{\bowtie} inactive\_M[5962] \\
&\underset{\{*\}}{\bowtie} active\_X[60] \underset{\{*\}}{\bowtie} inactive\_X[5962] \underset{\{*\}}{\bowtie} enzyme[6022]
\end{aligned}
$$

Bibliography

[1] A. Auger, P. Chatelain, and P. Koumoutsakos. R-leaping: accelerating the stochastic simulation algorithm by reaction leaps. *The Journal of Chemical Physics*, 125(8):084103, 2006.

[2] S. Balsamo, G. Dei Rossi, and A. Marin. Lumping and reversed processes in cooperating automata. In *Analytical and Stochastic Modeling Techniques and Applications,* LNCS 7314, pages 212–226. Springer Berlin Heidelberg, 2012.

[3] S. Balsamo, G. Dei Rossi, and A. Marin. Cooperating stochastic automata: approximate lumping a reversed process. In *International Symposium on Computer and Information Sciences*, pages 131–141. Springer London, 2013.

[4] J. Bezdek, R. Hathaway, M. Sabin, and W. Tucker. Convergence theory for fuzzy c-means: Counterexamples and repairs. *IEEE Trans. Syst. Man Cybern.*, 17(5):873–877, 1987.

[5] C. Bishop. *Pattern Recognition and Machine Learning*, volume 4. Springer, 1st edition, 2006.

[6] P. Buchholz. Exact and ordinary lumpability in finite Markov chains. *Journal of Applied Probability*, 31(1):59–75, 1994.

[7] K. Burrage, T. Tian, and P. Burrage. A multi-scaled approach for simulating chemical reaction systems. *Progress in Biophysics and Molecular Biology*, 85(2-3):217–234, 2004.

[8] A. Bušić and J. Fourneau. Bounds based on lumpable matrices for partially ordered state space. In *ICST Workshop on Tools for Solving Markov Chains*. ACM, 2006.

[9] X. Cai and J. Wen. Efficient exact and K-skip methods for stochastic simulation of coupled chemical reactions. *The Journal of Chemical Physics*, 131(6):064108, 2009.

[10] X. Cai and Z. Xu. K-leap method for accelerating stochastic simulation of coupled chemical reactions. *The Journal of Chemical Physics*, 126(7):074102, 2007.

[11] M. Calder, S. Gilmore, and J. Hillston. Modelling the influence of RKIP on the ERK signalling pathway using the stochastic process algebra PEPA. In *Transactions on Computational Systems Biology VII,* LNCS 4230, pages 1–23. Springer Berlin Heidelberg, 2006.

[12] Y. Cao, D. T. Gillespie, and L. Petzold. Accelerated stochastic simulation of the stiff enzyme-substrate reaction. *The Journal of Chemical Physics*, 123(14):144917–12, 2005.

[13] Y. Cao, D. T. Gillespie, and L. Petzold. Multiscale stochastic simulation algorithm with stochastic partial equilibrium assumption for chemically reacting systems. *Journal of Computational Physics*, 206(2):395–411, 2005.

[14] Y. Cao, D. T. Gillespie, and L. Petzold. Efficient step size selection for the tau-leaping simulation method. *The Journal of Chemical Physics*, 124(4):044109, 2006.

[15] Y. Cao, D. T. Gillespie, and L. R. Petzold. Avoiding negative populations in explicit Poisson tau-leaping. *The Journal of Chemical Physics*, 123(5):054104, 2005.

[16] Y. Cao, D. T. Gillespie, and L. R. Petzold. The slow-scale stochastic simulation algorithm. *The Journal of Chemical Physics*, 122(1):14116, 2005.

[17] Y. Cao, D. T. Gillespie, and L. R. Petzold. Adaptive explicit-implicit tau-leaping method with automatic tau selection. *The Journal of Chemical Physics*, 126(22):224101, 2007.

[18] Y. Cao, H. Li, and L. Petzold. Efficient formulation of the stochastic simulation algorithm for chemically reacting systems. *The Journal of Chemical Physics*, 121(9):4059–4067, 2004.

[19] Y. Cao and L. Petzold. Trapezoidal tau-leaping formula for the stochastic simulation of biochemical systems. In *Foundations of Systems Biology in Engineering*, pages 149–152, 2005.

[20] Y. Cao and L. Petzold. Accuracy limitations and the measurement of errors in the stochastic simulation of chemically reacting systems. *Journal of Computational Physics*, 212(1):6–24, 2006.

[21] Y. Cao and L. Petzold. Slow Scale Tau-leaping Method. *Computer Methods in Applied Mechanics and Engineering*, 197(43-44):3472–3479, 2008.

[22] A. Chatterjee, D. G. Vlachos, and M. A. Katsoulakis. Binomial distribution based tau-leap accelerated stochastic simulation. *The Journal of Chemical Physics*, 122(2):024112, 2005.

[23] F. Ciocchetta and J. Hillston. Bio-PEPA: A framework for the modelling and analysis of biological systems. *Theoretical Computer Science*, 410(33-34):3065–3084, 2009.

[24] A. Clark, A. Duguid, S. Gilmore, and M. Tribastone. Partial evaluation of PEPA models for fluid-flow analysis. In *Computer Performance Engineering (EPEW'08),* LNCS 5261, volume 5261, pages 2–16. Springer, 2008.

[25] A. Clark, S. Gilmore, M. Guerriero, and J. Hillston. Conservation of mass analysis for Bio-PEPA. In *Practical Applications of Stochastic Modelling,* ENTCS 296, pages 107–126, 2013.

[26] P. Courtois. Decomposability, instabilities, and saturation in multiprogramming systems. *Communications of the ACM*, 18(7):371–377, 1975.

[27] P. Courtois. Error analysis in nearly-completely decomposable stochastic systems. *Econometrica: Journal of the Econometric Society*, 43(4):691–709, 1975.

[28] D. Daly, P. Buchholz, and W. H. Sanders. Bound-preserving composition for Markov reward models. In *Quantitative Evaluation of Systems*, pages 243–252. IEEE Computer Society, 2006.

[29] K. Deng, P. Mehta, and S. Meyn. Optimal Kullback-Leibler aggregation via spectral theory of Markov chains. *IEEE Transactions on Automatic Control*, 56(12):2793–2808, 2011.

[30] K. Deng, Y. Sun, P. Mehta, and S. Meyn. An information-theoretic framework to aggregate a Markov chain. In *American Control Conference*, pages 731–736. IEEE Press, 2009.

[31] P. Deuflhard, W. Huisinga, A. Fischer, and C. Schütte. Identification of almost invariant aggregates in reversible nearly uncoupled Markov chains. *Linear Algebra and its Applications*, 315(1-3):39–59, 2000.

[32] R. Diestel. *Graduate Texts in Mathematics: Graph Theory*. Springer, 3rd edition, 2005.

[33] J. Fourneau, M. Lecoz, and F. Quessette. Algorithms for an irreducible and lumpable strong stochastic bound. *Linear Algebra and its Applications*, 386:167–185, 2004.

[34] G. Franceschinis and R. Muntz. Bounds for quasi-lumpable Markov chains. *Performance Evaluation*, 20(1-3):223–243, 1994.

[35] G. Franceschinis and R. Muntz. Computing bounds for the performance indices of quasi-lumpable stochastic well-formed nets. *IEEE Transactions on Software Engineering*, 20(7):516–525, 1994.

[36] G. Froyland. Statistically optimal almost-invariant sets. *Physica D: Nonlinear Phenomena*, 200(3-4):205–219, 2005.

[37] B. Furht and A. Escalante. *Handbook of Cloud Computing*. Springer, 2010.

[38] M. Gibson. *Computational Methods for Stochastic Biological Systems*. PhD thesis, California Institute of Technology, 2000.

[39] M. Gibson and J. Bruck. Efficient exact stochastic simulation of chemical systems with many species and many channels. *The Journal of Physical Chemistry*, 104(9):1876–1889, 2000.

[40] D. T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *The Journal of Physical Chemistry*, 81(25):2340–2361, 1977.

[41] D. T. Gillespie. Approximate accelerated stochastic simulation of chemically reacting systems. *The Journal of Chemical Physics*, 115(4):1716–1733, 2001.

[42] S. Gilmore, J. Hillston, and M. Ribaudo. An efficient algorithm for aggregating PEPA models. *IEEE Transactions on Software Engineering*, 27(5):449–464, 2001.

[43] A. Goldbeter. A minimal cascade model for the mitotic oscillator involving cyclin and cdc2 kinase. *Proceedings of the National Academy of Sciences of the United States of America*, 88(20):9107–9111, 1991.

[44] N. Götz, U. Herzog, and M. Rettelbach. Multiprocessor and distributed system design: The integration of functional specification and performance analysis using stochastic process algebras. In *Performance Evaluation of Computer and Communication Systems*, pages 121–146. Springer-Verlag, 1993.

[45] O. Häggström. *Finite Markov Chains and Algorithmic Applications*. Cambridge University Press, 2002.

[46] D. Hartfiel. On the structure of stochastic matrices with a subdominant eigenvalue near 1. *Linear Algebra and its Applications*, 272(1-3):193–203, 1998.

[47] R. Hayden and J. Bradley. A fluid analysis framework for a Markovian process algebra. *Theoretical Computer Science*, 411(22-24):2260–2297, May 2010.

[48] J. Hillston. *A Compositional Approach to Performance Modelling*. Cambridge University Press, 1996.

[49] J. Hillston. Fluid flow approximation of PEPA models. In *Quantitative Evaluation of Systems*, pages 33–42. IEEE Computer Society, 2005.

[50] J. Hillston and L. Kloul. An efficient Kronecker representation for PEPA models. In *Joint International Workshop on Process Algebra and Probabilistic Methods, Performance Modeling and Verification*, pages 120–135. Springer-Verlag, 2001.

[51] J. Himmelspach and A. M. Uhrmacher. The JAMES II Framework for Modeling and Simulation. *2009 International Workshop on High Performance Computational Systems Biology*, pages 101–102, 2009.

[52] D. Hume. Probability in transcriptional regulation and its implications for leukocyte differentiation and inducible gene expression. *Blood*, 96(7):2323–2328, 2000.

[53] M. Jacobi. A robust spectral method for finding lumpings and meta stable states of non-reversible Markov chains. *Electronic Transactions on Numerical Analysis*, 37(1):296–306, 2010.

[54] A. Jensen. Markoff chains as an aid in the study of Markoff processes. *Skandinavisk Aktuarietidskrift*, 36:87–91, 1953.

[55] J. Kemeny and J. Snell. *Finite Markov Chains*. Springer, 1976.

[56] A. M. Kierzek. STOCKS: STOChastic Kinetic Simulations of biochemical systems with Gillespie algorithm. *Bioinformatics*, 18(3):470–481, 2002.

[57] L. Kleinrock. *Queueing systems - Volume2: Computer applications*. John Wiley & Sons Inc, 1976.

[58] M. Kwiatkowska, G. Norman, and D. Parker. PRISM: probabilistic model checking for performance and reliability analysis. *ACM SIGMETRICS Performance Evaluation Review*, 36(4):40–45, 2009.

[59] M. Kwiatkowski and I. Stark. The continuous π-calculus: A process algebra for biochemical modelling. In *Computational Methods in Systems Biology*, pages 103–122. Springer-Verlag, 2008.

[60] K. Larsen and A. Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94(1):1–28, 1991.

[61] S. Lavenberg. *Computer Performance Modeling Handbook*. Academic Press, Inc., 1983.

[62] P. L'Ecuyer. TestU01: A C library for empirical testing of random number generators. *ACM Transactions on Mathematical Software*, 33(4):1–40, 2007.

[63] P. L'Ecuyer and E. Buist. Simulation in java with SSJ. *Proceedings of the Winter Simulation Conference 2005*, pages 611–620, 2005.

[64] H. Li and L. Petzold. Logarithmic direct method for discrete stochastic simulation of chemically reacting systems. *Technical report, Department of Computer Science, University of California, Santa Barbara*, 2006.

[65] U. Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4):395–416, 2007.

[66] J. Malik and J. Shi. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000.

[67] M. Matsumoto and T. Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation*, 8(1):3–30, 1998.

[68] H. McAdams and A. Arkin. Stochastic mechanisms in gene expression. *Proceedings of the National Academy of Sciences of the United States of America*, 94(3):814–819, 1997.

[69] J. M. McCollum, G. D. Peterson, C. D. Cox, M. L. Simpson, and N. F. Samatova. The sorting direct method for stochastic simulation of biochemical systems with varying reaction execution behavior. *Computational Biology and Chemistry*, 30(1):39–49, 2006.

[70] M. Meilă and J. Shi. A random walks view of spectral segmentation. In *AI and Statistics (AISTATS)*, pages 8–11. Citeseer, 2001.

[71] C. D. Meyer. Stochastic complementation, uncoupling Markov chains, and the theory of nearly reducible systems. *SIAM Review*, 31(2):240–272, 1989.

[72] D. Milios and S. Gilmore. Compositional approximate Markov chain aggregation for PEPA models. In *Computer Performance Engineering (EPEW'12),* LNCS 7587, pages 96–110. Springer, 2013.

[73] D. Milios and S. Gilmore. Markov chain simulation with fewer random samples. In *Practical Applications of Stochastic Modelling,* ENTCS 296, pages 183–197. Elsevier, 2013.

[74] I. Mitrani. *Probabilistic Modelling*. Cambridge University Press, 1st edition, 1998.

[75] E. Mjolsness, D. Orendorff, P. Chatelain, and P. Koumoutsakos. An exact accelerated stochastic simulation algorithm. *The Journal of Chemical Physics*, 130(14):144110–14, 2009.

[76] M. Molloy. Performance analysis using stochastic Petri nets. *IEEE Transactions on Computers*, 31(9):913–917, 1982.

[77] B. Nadler, S. Lafon, R. Coifman, and I. Kevrekidis. Diffusion maps, spectral clustering and eigenfunctions of Fokker-Planck operators. *Advances in Neural Information Processing Systems*, 18(1):955–962, 2006.

[78] A. Ng, M. Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm. *Advances in Neural Information Processing Systems*, 14(1):849–856, 2001.

[79] J. R. Norris. *Markov chains (Cambridge Series in Statistical and Probabilistic Mathematics).* Cambridge University Press, 1997.

[80] G. Plotkin. A Structured Approach to Operational Semantics. Technical report, Computer Science Department, Aarhus University, 1981.

[81] P. Pokarowski. Uncoupling measures and eigenvalues of stochastic matrices. *Journal of Applied Analysis*, 4(2):259–267, 1998.

[82] A. Pourranjbar, J. Hillston, and L. Bortolussi. Don't Just Go with the Flow: Cautionary Tales of Fluid Flow Approximation. In *Computer Performance Engineering (UKPEW'12),* LNCS 7587, pages 156–171. Springer, 2013.

[83] J. Puchalka and A. M. Kierzek. Bridging the gap between stochastic and deterministic regimes in the kinetic simulations of the biochemical reaction networks. *Biophysical Journal*, 86(3):1357–72, 2004.

[84] R. Ramaswamy, N. González-Segredo, and I. F. Sbalzarini. A new class of highly efficient exact stochastic simulation algorithms for chemical reaction networks. *The Journal of Chemical Physics*, 130(24):244104, 2009.

[85] 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.

[86] M. Rathinam, L. 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):12784–94, 2003.

[87] M. Rein. The partial-equilibrium approximation in reacting flows. *Physics of Fluids*, 4(1):873–886, 1992.

[88] T. Runolfsson and Y. Ma. Model reduction of nonreversible Markov chains. In *IEEE Conference on Decision and Control*, pages 3739–3744. IEEE, 2008.

[89] C. Sauer and K. Chandy. *Computer Systems Performance Modeling*. Prentice-Hall, 1981.

[90] M. Smith. *Stochastic Abstraction of Programs : Towards Performance-Driven Development*. PhD thesis, University of Edinburgh, 2010.

[91] M. Smith. Compositional abstractions for long-run properties of stochastic systems. In *Quantitative Evaluation of Systems*, pages 223–232. IEEE Computer Society, 2011.

[92] D. Wilkinson. *Stochastic Modelling for Systems Biology*. Chapman & Hall / CRC, 2006.

[93] I. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann Publishers, 2005.