Propagation Models for Biochemical Reaction Networks

THÈSE N^{O} 5088 (2011)

PRÉSENTÉE LE 4 NOVEMBRE 2011

À LA FACULTÉ INFORMATIQUE ET COMMUNICATIONS

LABORATOIRE DE MODÈLES ET THÉORIE DE CALCULS

PROGRAMME DOCTORAL EN INFORMATIQUE, COMMUNICATIONS ET INFORMATION

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES

PAR

Maria-Emanuela-Canini MATEESCU

acceptée sur proposition du jury:

Prof. P. Ienne, président du jury
Prof. T. Henzinger, directeur de thèse
Prof. M. Kwiatkowska, rapporteur
Prof. H. Köppl, rapporteur
Dr V. Wolf, rapporteur

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Suisse
2011

To Irina Athanasiu, who deeply cared about her students...

Abstract

In this thesis we investigate different ways of approximating the solution of the *chemical master equation* (CME). The CME is a system of differential equations that models the stochastic transient behaviour of *biochemical reaction networks*. It does so by describing the time evolution of probability distribution over the states of a Markov chain that represents a biological network, and thus its stochasticity is only implicit. The transient solution of a CME is the vector of probabilities over the states of the corresponding Markov chain at a certain time point t, and it has traditionally been obtained by applying methods that are general to continuous-time Markov chains: uniformization, Krylov subspace methods, and general ordinary differential equation (ODE) solvers such as the fourth order Runge-Kutta method.

Even though biochemical reaction networks are the main application of our work, some of our results are presented in the more general framework of *propagation models* (PM), a computational formalism that we introduce in the first part of this thesis. Each propagation model N has two associated propagation processes, one in discrete-time and a second one in continuous-time. These propagation processes propagate a generic mass through a discrete state space. For example, in order to model a CME, N propagates probability mass. In the discrete-time case the propagation is done step-wise, while in the continuous-time case it is done in a continuous flow defined by a differential equation. Again, in the case of the chemical master equation, this differential equation is the equivalent of the chemical master equation itself where probability mass is propagated through a discrete state space. Discrete-time propagation processes can encode methods such as the uniformization method and the fourth order Runge-Kutta integration method that we have mentioned above, and thus by optimizing propagation algorithms we optimize both of these methods simultaneously.

In the second part of our thesis, we define stochastic hybrid models that approximate the stochastic behaviour of biochemical reaction networks by treating some variables of the system deterministically. This deterministic approximation is done for species with large populations, for which stochasticity does not play an important role. We propose three such hybrid models, which we introduce from the coarsest to the most refined one: (i) the first one replaces some variables of the system with their overall expectations, (ii) the second one replaces some variables of the system with their expectations conditioned on the values of the stochastic variables, (iii) and finally, the third one, splits each variable into a stochastic part (for low valuations) and a deterministic part (for high valuations), while tracking the conditional expectation of the deterministic part. For each of these algorithms we give the corresponding propagation models that propagate not only probabilities but also the respective continuous approximations for the deterministic variables.

Keywords: *Markov chains; chemical master equation; biochemical reaction networks; numerical solutions; hybrid methods; probabilistic verification;*

Résumé

Ces travaux de thèse sont consacrés à l'étude des différentes façons d'estimation de la solution de *l'équation maîtresse chimique* (EMC). L'équation maîtresse chimique est un système d'équations différentielles stochastiques qui décrit le comportement transitoire des *réseaux de réactions biochimiques*. Certaines des techniques que nous proposons ne sont pas strictement liés à des réseaux de réaction biochimique et à leur EMC, mais cette équation reste le principal objectif tout au long de notre travail. L'EMC décrit l'évolution avec le temps des distributions de probabilité dans une chaîne de Markov qui représente un réseau biologique, et donc sa stochasticité n'est qu'implicite. Sa solution transitoire a traditionnellement été obtenue en appliquant des méthodes qui sont générales à des chaînes de Markov: la méthode de l'uniformisation, des méthodes de sous-espace Krylov, ou des méthodes générales aux équations différentielles ordinaires, comme par exemple, la méthode Runge-Kutta d'ordre 4.

Cette thèse est divisée en deux parties. Dans la première partie, nous définissons le formalisme de calcul des *modèles de propagation* (MP) et nous donnons des algorithmes qui estiment leur solution. A chaque modèle de propagation N nous associons deux processus de propagation, l'un en temps discret et l'autre en temps continu. Ces processus propagent de la masse à travers un espace discret d'états de N. Cette masse de propagation peut être de tout type et si, par exemple, le modèle de propagation représente une ECM, alors N propage des probabilités. Dans le cas du temps discret, la propagation est faite par étapes, tandis que dans le cas du temps continu, elle est faite dans un flux continu défini par une équation différentielle. Encore une fois, dans le cas de l'EMC, cette équation différentielle serait l'équivalent de l'équation maîtresse chimique elle-même où la masse de probabilité est propagée à travers un espace d'états discret. MP-s peuvent coder des méthodes telles que la méthode de l'uniformisation et la méthode de l'intégration de Runge-Kutta mentionnée

ci-dessus, et donc par l'optimisation des algorithmes MP nous optimisons ces deux méthodes simultanément.

Dans la deuxième partie, nous définissons des modèles stochastiques hybrides qui estiment le comportement stochastique de réseaux de réactions biochimiques par le traitement déterministe de certaines variables du système. Cette approximation déterministe est faite pour les espèces avec de grandes populations pour lesquelles la stochasticité ne joue pas un rôle important. Nous proposons trois de ces modèles hybrides présentés de la plus grossière à la plus raffinée: (i) la première remplace certaines variables du système avec leurs espérances globales; (ii) la seconde remplace certaines variables du système avec leurs espérances conditionnées par la valeur des variables stochastiques; (iii) la troisième divise chaque variable dans une partie stochastique (pour les valeurs basses) et une partie déterministe (pour les valeurs hautes), tout en suivant l'espérance conditionnelle de la partie déterministe. Pour chacun de ces algorithmes, nous donnons les modèles de propagation correspondant qui ne propagent pas seulement des probabilités, mais aussi les approximations continues respectives des variables discrètes.

Mots-clés: *chaînes de Markov; équation maîtresse chimique; réseaux de réaction biochimique; solutions numériques; méthodes hybrides; vérification probabiliste;*

Acknowledgments

First of all, I am thankful to the members of my jury, for taking the time to carefully read this thesis and for their insightful and useful feedback.

I have joined the EPFL doctoral program without even knowing that the field of model checking existed. However, Tom's class on Computer Aided Verification fascinated me right from the beginning: a mixture of my two favourite subjects (mathematics and computer science) offered with the clarity and sharpness of Tom's presentation style. After attending this class, I simply could not say no to joining the Models and Theory of Computation (MTC) laboratory as a PhD student. Thanks to Jasmin and Nir, I have then more specifically started to work on problems with applications in biology, and especially since Verena joined our group my research focused on numerical solutions of the Chemical Master Equation. Verena has strongly influenced the outcome of my thesis, for which I'm very grateful. I have learned many things from her, and what I value most is that she taught me how to approach hard problems by taking small steps in solving them. Fred is another person that had an important influence on my research, and he has done so out of a pure interest in sciences, which turned him from a friend to a co-author. I have learned from him how to look at the heart of a problem, and he also taught me a whole lot about programming.

Tom is the best advisor I could have chosen for my PhD. He taught me how to feel if a formalism is right or not, how to make everything fit in a big picture, how to be clear, and many other things. He has wisely chosen very special people to be part of our group. Most of the time at EPFL I have spent with my office-mate, Grégory. Someone once told me: "If you have a problem, see Grégory." That was as accurate as it can get, thank you Grégory for your for the good times in the office, for your kindness, wisdom and support. As I have said, all colleagues I had in MTC are great people, and I consider myself very lucky to have gained the valuable friendship of Laura, Laurent, Dietmar, Dejan, Barbara, Tatjana and Vasu. I am also very grateful to Fabien and Sylvie who

made all “logistics” run so smoothly that for years I almost forgot that such problems can ever exist.

EPFL proved to have a very diverse and open community where I made friends from all over the world. Karin literally saved me one day when she showed me a group of about ten people and told me: now you can call all of them your friends. I appreciated her trying to be so nice, but I never thought she was actually telling the truth. One by one I got to know all of them, and then many others, and each one had something to share: Nino - bad but lovely sense of humour, Harm - making it seem so easy, Zafer - still owes me a violin concert, Klaske - maybe the hardest one to win over, German - the great apple break conspiracies, Veronica - best motivator ever, Denisa - making magic happen, Ghid - devotion and perfect replacement for myself, Luigi - being young, Oli - being very young, Lorenzo - bicycle mania, Nigel Farquhar-Bennett - unstoppable curiosity, Wojciech & co - the movie makers, Jasper - most committed father, Ingmar - chocolate, sports, knowledge and friendship mania, Ashutosh - seeking for trouble but in a good way, Maya - never giving up, Tobi - being a gentleman, Jim - road opener, Bram - keeping the flag up, Albrecht - laughing, Andrea - probability theory and aiming high, Ashkan - hugs and hard metal, Ben - balloons, Bertrand - tart perfection and a bit of math, Pam - loving nature, Gregg - priorities expert, Davor - love-hate relationship, Hamed - there when you need him, Voja - simply friendly, Dan - encouraging, Lukas - natural, Nicolas - taking me very high up the mountains, Adriaan - best teaser, Willem-Jan - piano playing, Lina - giving it all, Tanja - great pies and great smile, Claudia - bicycle dancing, Alex Susu - the right words at the right time, Viktor - great teacher, Pieter - thinking abstract and, Razvan - saved me when I just got to Lausanne. I know this is a very long list, and it is so because I’m one lucky person, and each of these people made my PhD studies brighter and brighter still. Among them I must make a special remark about Denisa, Dan and Ghid, whose advices and support while I was preparing my thesis as well as my private thesis exam were what anyone would have dreamed for. And among them I must add a special thank you to Denisa for being like a sister to me, and without whom I might never have called Lausanne to be home.

I have to mention that I have gained a lot from the doctoral school or master courses at EPFL, it is here for example that I have refreshed my notions of probability theory. I took on two internships while being a PhD student, both of which have broaden my horizons in valuable ways, and also allowed me to meet very motivated people at ETHZ in Uwe Sauer’s group and in Google Zurich. I am thankful for both of these opportunities. My PhD studies were

funded through an SNSF grant which allowed me to attend numerous conferences, summer schools and workshops, from which I have learned a lot, especially in the field of systems biology. Among these meetings I must mention the q-bio summer school 2010, where I have made true friends and where the big picture of the quantitative biology field started to take shape in my mind.

EPFL has a vast collection of associations among which I have mostly been involved with A/RO, GSA, ACIDE and Polyathlon. All of these associations have offered a lot to me, in terms of people that I have met and events that I attended and even management skills that I have acquired. Also I am thankful for the valuable experience that I have gained while being a first-aid volunteer on campus.

Without knowing so many fine Romanians in EPFL I would have felt alienated on foreign land. But together with Diana, Mihai, Iuli, Nicu, Cristi, Andrea, Veronica, Alex, Roxana, Bogdan, and Oana to name just a few, we made our own little Romania here in EPFL. Diana was especially important in this mission as it was mostly the two of us that put together A/RO, the Romanian association. Nicu and Iuli also deserve a special note, because they have listened to a lot of my mumbling about life and other things. Thank you!

I must also mention the Lyon family who welcomed me to Switzerland as if I was their daughter, even though I was a stranger to them, as well as Denise Poenaru who had encouraged me to embrace piano playing.

I am also grateful for their continuous support to my friends back in Romania, especially: Vivi, Irina, Dan, Irina, and Mihaela. And speaking of people that I left in Romania, I should mention Freescale, the company where I was working before joining a PhD program. The time that I spent there and the people that I got to meet in that period will always stay very dear in my heart. And it is Freescale that made me interested in hardware, which made me interested in computer aided verification, which lead me to this thesis. Also, during my studies at "Politehnica" University of Bucharest was when I first grew as an engineer and even as a scientist. I often think about Irina Athanasiu, who passed away in 2006. She was my master diploma supervisor, and I thank her for being a role-model to me, for making me think, for making me dream and dare.

And finally, there is my warm and loving family who always trusted in me, and was always there to celebrate a success, or to heal a wound. I am the luckiest person for such grandparents who always trusted in me, uncles together with their families who always looked after me, a father who made me see mathe-

matics everywhere, a brother whose encouragements while writing parts of this thesis were priceless, Coca and Maria who always made Christmas Christmas, a mother who is not with us any more, but whose memory is ever present and who taught me how to count, without which we must agree this thesis would not have been possible. And then, there is the cherry on my cake, Jérôme, the man who won my heart immediately and who would better keep it for a long time. Look, Jérôme, your (little) girl has finished writing!

Previous Publications

The research presented in Section 4.2 appeared as T. A. Henzinger, and M. Mateescu, V. Wolf, CAV '09. "Sliding Window Abstraction for Infinite Markov Chains", in the *Proceedings of the 21st International Conference on Computer Aided Verification (CAV 2009)*, Lecture Notes in Computer Science 5643, 2009; and as "Solving the chemical master equation using sliding windows", together with R. Goel, in the *BMC Systems Biology journal*, 4:42, 2010.

The research presented in Section 5.1 appeared as F. Didier, T.A. Henzinger and M. Mateescu, and V. Wolf, "Approximation of Event Probabilities in Noisy Cellular Processes", in the *Proceedings of the 7th International Conference on Computational Methods in Systems Biology (CMSB 2009)*, Lecture Notes in Computer Science 5688, Springer, 2009; and in the *Theoretical Computer Science journal*, 412, 2011.

The research presented in Chapter 3 appeared as F. Didier, T.A. Henzinger, M. Mateescu, and V. Wolf, "Fast Adaptive Uniformization of the Chemical Master Equation", in the *Proceedings of the first International Workshop on High Performance Computational Systems Biology (HiBi 2009)*, IEEE Computer Society, 2009; and in *Systems Biology, IET journal*, 4:6, 2010. These publications do not use the propagation models formalism.

The research presented in Chapter 8 appeared as T.A. Henzinger, L. Mikeev, M. Mateescu, and V. Wolf, "Hybrid Numerical Solution of the Chemical Master Equation", in the *Proceedings of the 8th International Conference on Computational Methods in Systems Biology, (CMSB 2010)*, ACM, 2010.

The research presented in Chapter 11 appeared as F. Didier, T.A. Henzinger, M. Mateescu, and V. Wolf, "SABRE: A Stochastic Analysis Tool for Biochemical Reaction Networks", in the *Proceedings of the 7th International Conference on Quantitative Evaluation of SysTems (QEST 2010)*, IEEE Computer Society, 2010.

The research presented in Chapter 9 appeared as T.A. Henzinger, and M. Mateescu, "Tail Approximation for the Chemical Master Equation", in the *Proceedings of the 8th International Workshop on Computational Systems Biology (WCSB 2011)*.

And finally the propagation models formalism was introduced in T.A. Henzinger, and M. Mateescu, "Propagation Models for Computing Biochemical Reaction Networks", int the *Proceedings of the 9th International Conference on Computational Methods in Systems Biology (CMSB 2011) - to appear*.

Contents

Chapter 1

Introduction

1.1 Motivation

Biochemical reaction networks are widely used within systems biology to describe biological processes. They consist of a list of chemical reactions with associated rates, e.g. $V + W \xrightarrow{10} C$, where the rate 10 characterizes the affinity that molecules V and W have for each other. Such networks are used in mainly three kinds of analysis: deterministic average behaviour, stochastic simulation and, the main subject of this thesis, numerical analysis of the probability distributions over possible states[1] of the system.

The *average behaviour analysis* is constructing a system of ordinary differential equations called the *reaction rate equation* (RRE) that approximates the time evolution of the concentrations of certain molecules in a biological compartment. This macroscopic model is based on the theory of chemical kinetics and assumes that the concentrations of chemical species in a well-stirred system change deterministically and continuously in time. It provides an appropriate description of a chemically reacting system as long as the number of molecules of the chemical species are large.

However, molecular noise, which arises from the randomness of the discrete events in the cell, significantly influences fundamental biological processes such as gene expression [27, 108], decisions of the cell fate [5, 76], and circadian oscillations [7, 43]. This is especially the case, when the chemical populations of some species are low (e.g., in living cells we have a single DNA molecule, tens or a few hundreds of RNA or protein molecules). In this case, the underlying

[1] As shown later, a state of a biochemical reaction network is described by the copy numbers for each molecule species.

assumptions of the RRE approach are violated and a more detailed model is necessary [32, 79, 87, 93, 107].

During the last decade, stochastic models with discrete state space have seen a growing interest because they provide appropriate descriptions of systems that are subject to molecular noise, therefore addressing the limitations of the deterministic models. The theory of *stochastic chemical kinetics* provides an appropriate description by means of a discrete-state Markov process, that is, a continuous-time Markov chain (CTMC) that represents the chemical populations as random variables [38, 39]. In the thermodynamic limit (when the number of molecules and the volume of the system approach infinity) the Markov model and the macroscopic RRE description are equal [70], and as already mentioned, the RRE approach can only be used to approximate the Markov chain if all populations are large.

The evolution of the Markov chain is given by a system of linear ordinary differential equations, known as the *chemical master equation*(CME). A single equation in the CME describes the time derivative of the probability of a certain state at all times $t \geq 0$. Thus, the solution of the CME is the probability distribution over all states of the Markov chain at a particular time t, that is, the transient state probabilities at time t. The solution of the CME can then be used to derive measures of interest such as the distribution of switching delays [78], the distribution of the time of DNA replication initiation at different origins [86], or the distribution of gene expression products [110]. Moreover, many parameter estimation methods require the computation of the posterior distribution because means and variances do not provide enough information to calibrate parameters [55].

The more detailed description of chemical reactions using the CME comes at a price of significantly increased computational complexity because the underlying state space is usually very large or even infinite. Therefore, *stochastic simulation* is in widespread use, because it allows to generate random trajectories of the model while requiring only little memory. Estimates of the measures of interest can be derived once the number of trajectories is large enough to achieve the desired statistical accuracy. However, the main drawback of simulative solution techniques is that a large number of trajectories is necessary to obtain reliable results. For instance, in order to halve the confidence interval of an estimate, four times more trajectories have to be generated. Consequently, often stochastic simulation is only feasible with a very low level of confidence in the accuracy of the results.

1.2 Numerical Transient Solutions

The focus of this thesis is on numerical algorithms for the transient solution of the CME of biochemical reaction networks.

Example 1.1. *Consider a biochemical reaction network with two species, V and W, whose stochastic behaviour is given by a CTMC $\mathbf{X}(t) = (V(t), W(t))$, where $V(t)$ is a random variable that gives the number of molecules of type V that are present in the system at time t, and the same for $W(t)$. The CTMC $\mathbf{X}(t)$ is completely described by a graph whose edges are labelled by real valued rates. Let $\mathbf{s}'$ be a successor of the state $\mathbf{s}$ in this graph, and let the rate of jumping from $\mathbf{s}$ to $\mathbf{s}'$ be* 10*. Then, we have that*

$$Pr(\mathbf{X}(t + dt) = \mathbf{s}' \,|\, \mathbf{X}(t) = \mathbf{s}) = 10 \cdot dt,$$

for an infinitesimal value dt. Given an initial state $(\mathbf{X}(0) = \mathbf{y})$, and a time point $t \geq 0$, the transient analysis problem *for Markov chains asks that we compute the values:*

$$Pr(\mathbf{X}(t) = \mathbf{s} \,|\, \mathbf{X}(0) = \mathbf{y}) \,, \forall \mathbf{s} = (v, w).$$

Intuitively, computing this transient solution is the equivalent of propagating*, in small steps dt, the probability to be in state $\mathbf{s}$ times the probability to jump from state $\mathbf{s}$ to a successor of $\mathbf{s}'$, for all pairs of states and their successors. For example, at time t, state $\mathbf{s}$ propagates to $\mathbf{s}'$ the value val*

$$Pr(\mathbf{X}(t) = \mathbf{s} \,|\, \mathbf{X}(0) = \mathbf{y}) \cdot 10 \cdot dt =: val, \tag{1.1}$$

and so do, in parallel, all the other states of the Markov chain.

Traditionally, the CME has been solved using one of three classes of numerical methods that are general to the transient solution of any CTMC: uniformization methods, general ODE solvers such as Runge-Kutta fourth order, or Krylov subspace methods; and this thesis deals with optimizing the former two of these methods for the case of the chemical master equation.

For a CTMC $\{X(t), t \geq 0\}$ The *uniformization method* [105] is defining a uniformized discrete-time Markov chain (DTMC) $\{X^u(k), k = 0, 1, \ldots\}$ and a Poisson process $\left\{X^P(t), t > 0\right\}$ such that $X(t)$ is stochastically identical to $X^u(X^P(t))$. The Poisson process $X^P(t)$ can be interpreted as a clock that stochastically decides how many steps are taken by the uniformized discrete-

time Markov chain $X^u(t)$ in a time interval of length t, such that its stochastic behaviour matches the stochastic behaviour of $X(t)$. From a computational point of view, uniformization reduces the problem of integrating a continuous-time differential equation (the CME) to that of solving a discrete-time Markov chain through a series of matrix-vector products. A variant of uniformization, called *adaptive uniformization*, is based on the same principles, with the difference that the clock for the steps of the adaptively uniformized discrete-time Markov chain is given by a birth process, which is a more general jump process than the Poisson process.

The *fourth order Runge-Kutta method* is a classic integration method. It is used to integrate the CME by taking small time steps, thus, applying a direct discretization of the continuous-time. In each integration step, probability is pushed through the system according to the derivative of the probability mass function, as given by the CME.

1.3 Propagation Models

In the first part of this thesis we introduce the computational formalism of *propagation models*. A propagation model is a graph, whose nodes are called *states*, and whose edges have associated edge functions that describe how much mass is to be propagated from one state of the graph to another one. A propagation model accepts two semantics, one that defines a *discrete-time propagation processes* (DTPP), and a second one that defines a *continuous-time propagation processes* (CTPP). These processes spread *mass* through the propagation model's graph according to the valuations of the edge functions. The valuation of a propagation process at a certain time point is a mass vector over the states of the model (the nodes of the graph).

Example 1.2. *(cont.) For the simple CTMC $X(t)$ from Example 1.1, we define a propagation model which has the same state space as the Markov chain, and its* edge function *between states* $\mathbf{s}$ *and* $\mathbf{s}'$ *is equal to the propagated value val from Equation 1.1.*

After defining this formalism, we deal with the problem of evaluating a discrete or a continuous-time propagation processes at a certain time point, by designing different propagation algorithms. The basic discrete-time propagation algorithm is straightforward, as it simply involves a value iteration procedure, and the main challenge is in optimizing it for a good accuracy/performance

ratio. For this we rely on an *on-the-fly* construction of the graph and on a *threshold abstraction* that ignores states that hold insignificant mass.

For the continuous-time case, we first introduce the *sliding window method*, which is conceived strictly for propagation models that describe the CME of biochemical reaction networks. This is why we choose to present this method in the framework of *transition class models*, a formalism that is very well suited to describe biochemical reaction networks. This method computes an approximate solution of the CME by performing a sequence of local analysis steps. In each step, only a manageable subset of states is considered, representing a "window" into the state space. In subsequent steps, the window follows the direction in which the probability mass moves, until the time period of interest has elapsed. We construct the window based on a deterministic approximation of the future behaviour of the system by estimating upper and lower bounds on the populations of the chemical species. The local solutions can be computed using any numerical method for general Markov chains, such as those that we have already mentioned. In order to show the effectiveness of this approach, we apply it to several examples previously described in the literature. The experimental results show that the proposed method speeds up the analysis considerably, compared to a global analysis, while still providing high accuracy.

We continue with algorithms that evaluate general continuous-time propagation processes by using either the Runge-Kutta integration method or the uniformization method. Runge-Kutta method is a classic ODE solver, and thus it can be directly applied to our problem, because CTPPs are described by an ordinary differential equation. Uniformization, however, is specific to the transient analysis of continuous-time Markov chains and thus needs to be generalized in order to pass from CTMCs to CTPPs. The generalizations that we propose, both for standard and adaptive uniformization, are exact only for propagation models that have a certain linearity property that we discuss later, and approximative for other propagation models. However, it is important to note already that propagation models that correspond to a CME have this linearity property and thus our computational formalism is not introducing any additional approximations.

Solving the CTPP problem by using uniformization or Runge-Kutta methods can be seen as expressing these two methods in the framework of propagation models. As we have already presented concisely, both uniformization and Runge-Kutta integration involve propagating probability mass through the Markov chain graph: uniformization by computing the product of a matrix with a probability distribution vector, and Runge-Kutta by pushing probability, in

each integration step, from one state towards its neighbours as described by the CME. In particular, Runge-Kutta fourth order integration is pushing probability towards neighbours that are up to four edges away in the Markov chain graph. Therefore, expressing these algorithms in the framework of propagation models comes naturally.

Even though the propagation algorithms that we propose are not restricted to Markov chains, the main application of our work remains the chemical master equation. The advantage of formulating these two algorithms for Markov chains in the framework of propagation models is that optimizations proposed for propagation models can then be applied to both of these algorithms simultaneously. We have chosen to present case studies for *fast adaptive uniformization*, a method that results from the combination of discrete-time propagation processes with adaptive uniformization. These results show that threshold abstraction brings considerable benefits to the original adaptive uniformization algorithm at a small price in precision loss.

To summarize, sliding window is a method that reduces the state space of the CME directly in the continuous-time framework, and then it recurs to any numerical solver for the transient solution of Markov chains. On the other hand, the general propagation algorithms that we present first use uniformization or Runge-Kutta integration to reduce the continuous-time problem to a discrete-time problem and then they recur to the threshold abstraction for DTPP in order to reduce the space of the discrete-time process.

1.4 Hybrid Propagation Models

While in the first part of the thesis we mainly deal with algorithms for general propagation models, in the second part of the thesis we present algorithms that are optimized for biochemical reaction networks. We start from the observation that molecular noise, and stochastic effects, appear especially when some molecular species are present with significant probability in a low copy number, while for species that are present in a large copy number their expectation is providing enough information about the system. Based on this observation, the optimizations that we propose in the second part of this thesis take advantage of the fact that the average behaviour of species with large populations can be approximated by the reaction rate equation. Therefore, we construct hybrid stochastic models that mix the CME and the RRE, thus treating some variables of the system stochastically and some deterministically. When a propagation

process is used to describe a pure CME, it only propagates probabilities through the graph of the model. Here, for stochastic hybrid models, the propagation processes that we construct propagate both probabilities and expectations of the deterministic variables.

There are several ways one can combine these two equations, the CME and the RRE, and here we present three variants of this approach. Recall Example 1.1 that describes a system with two species V and W. For each of the three hybrid approaches, we define an aggregated problem that instead of asking for the computation of the probabilities $Pr(V(t) = v, W(t) = w)$, for all values v and w asks for the computation of aggregated probabilities, such as the probability for $V(t)$ to be v (independent of the value of $W(t)$). In addition, an aggregated problem also asks for the computation of expectations such as: $E[W(t) \mid V(t) = v]$.

This first method is the *naive hybrid stochastic method*. Continuing our toy example, consider that V is a species with low populations, while W is a species with high populations. The *naive aggregated solution* consists of the probabilities $Pr(V(t) = v)$ for all values v, as well as the expectations $E[W(t)]$. In order to approximate this naive aggregated solution, we construct two ODEs, one derived from the CME, which approximates the time derivative of $Pr(V(t) = v)$, and one derived from the RRE, which approximates the time derivative of $E[W(t)]$. These ODEs are interlinked because the value of the expectation of $W(t)$ might be used by the reduced CME and the probabilities of $V(t) = v$ might be used by the RRE equation, and thus the two equations update each other mutually.

The second method is the *hybrid stochastic method*. Here, we approximate a *conditional aggregated solution*, which requires the computation of the values $Pr(V(t) = v)$ and of the conditional expectations $E[W(t) \mid V(t) = v]$, for all values v. Again, we construct differential equations that approximate the evolution of the probabilities for low copy species and of the conditional expectations of large populations, and as in the naive case, these differential equations depend on each other.

Finally, the *tail approximation method* proposes the approximation of the *tail aggregated solution*. The tail aggregated solution does not depend on a separation of the species in low and high populations, as in the other two methods, but it splits the population domain of each species into a low and a high

part with respect to a boundary b. For our small example, the tail aggregated solution comprises of the following probabilities and conditional expectations:

$$
\begin{aligned}
&Pr(V(t) = v, W(t) = w)\,, && \forall v, w < b,\\
&Pr(V(t) = v, W(t) \geq b)\,,\ E[W(t)|W(t) \geq b, V(t) = v] && \forall v < b,\\
&Pr(V(t) \geq b, W(t) = w)\,, E[V(t)|V(t) \geq b, W(t) = w] && \forall w < b,\\
&Pr(V(t) \geq b, W(t) \geq b)\,,\ E[(V(t), W(t))|V(t) \geq b, W(t) \geq w].
\end{aligned}
$$

Connecting the differential equations that describe the time behaviour of these probabilities and of conditional expectations is technically difficult because the reduced CME needs more information about the states with large molecule numbers than the RRE is providing, e.g. it needs the probabilities of the states at the border b. We are addressing this issue by approximating the probabilities of the border states that are governed by the reaction rate equation with probabilities from a geometric distribution to which we apply a correction scheme.

For the first two methods we provide experiments done on biological case studies and the results support our theoretical conclusion that these approximations perform well in the case of biological reaction networks. For tail approximation, we only present results that demonstrate the accuracy of our heuristic for approximating border probabilities.

Chapter 2

Preliminaries

2.1 Biochemical Reaction Networks

We consider a fixed reaction volume with n different chemical species that is spatially homogeneous and in thermal equilibrium. Then, the state space of the system is $\mathbb{N}_0^n$. We assume that molecules collide randomly and that collisions may lead to chemical reactions. Formally, assume that the network consists of m different chemical reactions. Let $j \in \{1, \ldots, m\}$, and let the j-th reaction be given by the stoichiometric equation

$$k_1 T_1 + \ldots + k_n T_n \to \ell_1 T_1 + \ldots + \ell_n T_n$$

where for $i \in \{1, \ldots, n\}$ the symbol T_i refers to the i-th chemical species and the stoichiometric coefficients k_i, ℓ_i are non-negative integers that specify how many molecules of type i are consumed and how many are produced by the reaction, respectively. If $k_i > 0$ then the i-th species is called a reactant of the j-th reaction. In stoichiometric equations, terms with coefficient 0 are usually omitted and terms of the form $1T_i$ are abbreviated by T_i. The symbol $\emptyset$ abbreviates the case $0 = k_1 = \ldots = k_n$ or $0 = \ell_1 = \ldots = \ell_n$.

Example 2.1 (Gene expression)**.** *We consider a simple biochemical reaction network for transcription of a gene into messenger RNA (mRNA), and subsequent translation of the latter into proteins [108]. This reaction network involves three chemical species, namely, gene, mRNA, and protein. As only a single copy of the gene exists, a state of the system is uniquely determined by the number of mRNA and protein molecules. Therefore the state space of the model is $\mathbb{N}_0^2$ and a state is a pair (x_R, x_P). We assume that initially there are no mRNA molecules*

and no proteins in the system, i.e., $\mathbf{y} = (0, 0)$*. The following four types of reactions occur in the system, namely* $\emptyset \to mRNA$*,* $mRNA \to mRNA + P$*,* $mRNA \to \emptyset$*, and* $P \to \emptyset$*.*

2.2 Transition Class Models

This section introduces a high-level modeling formalism, called *Transition Class Models* (TCMs). TCMs provide a functional description of structured Markov chains with countably infinite state spaces, and have been used in queuing theory [106] and recently for stochastic models of coupled chemical reactions [99].

Consider a dynamical system with a finite number of discrete state variables such as the number of instances of some chemical species in a reaction volume. Assume that these variables change at discrete points in time. A *transition class* provides a rule for these changes and a function for the calculation of the state-dependent *transition rate* at which a state change occurs.

Definition 2.0.1. *A* transition class model *(TCM) is a pair* $\langle \mathcal{S}, \mathbf{y}, \{\mathcal{C}_1, \ldots, \mathcal{C}_m\}\rangle$*, where* $\mathcal{S}$ *is a countable set of states,* $\mathbf{y} \in \mathcal{S}$ *is an initial state and* $\{\mathcal{C}_1, \ldots, \mathcal{C}_m\}$ *is a finite set of transition classes. Each* transition class $\mathcal{C}$ *is a triple* (G, u, α) *with a* guard set $G \subseteq \mathcal{S}$*, an* update function $u : G \to \mathcal{S}$*, and a* rate function $\alpha : G \to \mathbb{R}_{>0}$*.*

The guard set G contains all states $\mathbf{s}$ for which a transition of class $\mathcal{C}$ is possible, and $u(\mathbf{s})$ is the target state of the transition. Each $\mathcal{C}$-transition also has an associated rate $\alpha(\mathbf{s})$ that depends on the current state $\mathbf{s}$.

For a biochemical reaction network, we define the j-th transition class $\mathcal{C}_j = (G_j, u_j, \alpha_j)$ such that

$$G_j = \{\mathbf{s} = (s_1, \ldots, s_n) \in \mathbb{N}_0^n \mid s_i \geq k_i, i \in \{0, 1, \ldots n\}\}, \tag{2.1}$$

$$u_j(\mathbf{s}) = \mathbf{s} + (\ell_1 - k_1, \ldots, \ell_n - k_n) = \mathbf{s} + \mathbf{d}_j, \tag{2.2}$$

$$\alpha_j(\mathbf{s}) = c_j \cdot \prod_{i=1}^{n} \binom{s_i}{k_i}. \tag{2.3}$$

We call $\mathbf{d}_j = (\ell_1 - k_1, \ldots, \ell_n - k_n)$ *the change vector* of the j-th reaction of the biochemical reaction network. The rate function α_j takes into account that the probability of a reaction of type j is proportional to the possible number of combinations of reactant molecules, i.e., if k_i molecules of type i are needed and the current number of molecules of type T_i is s_i then $\binom{s_i}{k_i}$ is the number of possible ways to choose k_i out of s_i. The rate constant $c_j > 0$ depends on

the temperature, the volume, and the micro-physical properties of the reactant species [39].

Example 2.2 (Gene expression continued.)**.** *Let $j \in \{1, \ldots, 4\}$ and let $c_j > 0$ be a constant. Transition class $\mathcal{C}_j = (G_j, u_j, \alpha_j)$ describes the j-th reaction type.*

- *We describe* gene transcription *by transition class $\mathcal{C}_1$, which increases the number of mRNA molecules by 1. Thus, $u_1(s_R, s_P) = (s_R + 1, s_P)$. This transition class is possible in all states, i.e., $G_1 = \mathcal{S}$. Transcription happens at the constant rate $\alpha_1(s_R, s_P) = c_1$, as only one reactant molecule (the gene) is available.*
- *We represent the* translation *of mRNA into protein by $\mathcal{C}_2$. A $\mathcal{C}_2$-transition is only possible if there is at least one mRNA molecule in the system. We set $G_2 = \{(s_R, s_P) \in \mathcal{S} \mid s_R > 0\}$ and $u_2(s_R, s_P) = (s_R, s_P + 1)$. Note that in this case mRNA is a reactant that is not consumed. The translation rate depends linearly on the number of mRNA molecules. Therefore, $\alpha_2(s_R, s_P) = c_2 \cdot s_R$.*
- Degradation *is modeled by $\mathcal{C}_3$ and $\mathcal{C}_4$. Hence, $G_3 = G_2$, $G_4 = \{(s_R, s_P) \in \mathcal{S} \mid s_P > 0\}$, $u_3(s_R, s_P) = (s_R - 1, s_P)$, and $u_4(s_R, s_P) = (s_R, s_P - 1)$. We set $\alpha_3(s_R, s_P) = c_3 \cdot s_R$ and $\alpha_4(s_R, s_P) = c_4 \cdot s_P$.*

2.2.1 Chemical Master Equation

A transition class model $M = \langle \mathcal{S}, \mathbf{y}, \{\mathcal{C}_1, \ldots, \mathcal{C}_m\} \rangle$ represents a time-homogeneous, discrete-state Markov process (CTMC, for short) $(\mathbf{X}(t))_{t \geq 0}$ with state space $\mathcal{S}$.[1]

The i-th entry of the random vector $\mathbf{X}(t) = (X_1(t), \ldots, X_n(t))$ represents the value of the i-th state variable. Let $\mathcal{C}_j = (G_j, u_j, \alpha_j)$, $1 \leq j \leq m$, and assume that at time $t \geq 0$ the process is in state $\mathbf{s} \in G_j$.

The probability of a transition of type $\mathcal{C}_j$ to occur in the next infinitesimal time interval $[t, t + dt)$, $dt > 0$ is given by

$$Pr(\mathbf{X}(t + dt) = u_j(\mathbf{s}) \mid \mathbf{X}(t) = \mathbf{s}) = \alpha_j(\mathbf{s}) \cdot dt + o(dt),$$

where $o(h)$ represents a function with $\lim_{h \downarrow 0} \frac{o(h)}{h} = 0$ and $o(0) = 0$.

For $\mathbf{s} \in \mathcal{S}$ we define the probability that $\mathbf{X}(t)$ is in state $\mathbf{s}$ at time t by $p_{\mathbf{s}}^{(t)} = Pr(\mathbf{X}(t) = \mathbf{s} \mid \mathbf{X}(0) = \mathbf{y})$. Note that it follows that $p_{\mathbf{y}}^{(0)} = 1$. Now recall that u_j is injective. To simplify our presentation, we define the set H_j as the

[1] When understandable from the context, we loosely refer to the stochastic process $(\mathbf{X}(t))_{t \geq 0}$ simply as $X(t)$. Similarly, in the case of discrete time we may loosely refer to processes $(\mathbf{X}(k))_{k \in \mathbb{N}_0}$ as $X(k)$.

set of all states $\mathbf{s}$ for which $u_j^{-1}(\mathbf{s})$ is defined, that is, the set of states that can be reached by a transition of type C_j. The *chemical master equation* (CME) describes the behaviour of $\mathbf{X}(t)$ by the differential equation [65]

$$\frac{d\boldsymbol{p}_{\mathbf{s}}^{(t)}}{dt} = \sum_{j:\mathbf{s}\in H_j} \alpha_j(u_j^{-1}(\mathbf{s})) \cdot \boldsymbol{p}^{(t)}(u_j^{-1}(\mathbf{s})) \tag{2.4}$$

$$- \sum_{j:\mathbf{s}\in G_j} \alpha_j(\mathbf{s}) \cdot p_{\mathbf{s}}^{(t)}. \tag{2.5}$$

Note that there exist pathological cases in which $\mathbf{X}(t)$ is not uniquely defined by M [58]. These cases are, however, not relevant for the application area that we consider here. We therefore assume that M is such that it uniquely defines a Markov process $\mathbf{X}(t)$.

In our subsequent presentation, a matrix description of the CME is more advantageous. It is obtained by defining the *infinitesimal generator matrix* $Q = (Q(\mathbf{s}, \mathbf{s}'))_{\mathbf{s},\mathbf{s}'\in\mathcal{S}}$ of $\mathbf{X}(t)$ by

$$Q(\mathbf{s}, \mathbf{s}') = \begin{cases} \alpha_j(\mathbf{s}) & \text{if } u_j(\mathbf{s}) = \mathbf{s}', \\ -\sum_{j=1}^{m} \alpha_j(\mathbf{s}) & \text{if } \mathbf{s} = \mathbf{s}', \\ 0 & \text{otherwise,} \end{cases}$$

where we assume a fixed enumeration of the state space. The row sums of the (possibly infinite) matrix Q are zero, and $\lambda_s = -Q(\mathbf{s}, \mathbf{s})$, the *exit rate* of state $\mathbf{s}$, is the reciprocal value of the average residence time in $\mathbf{s}$.

Let $T^{(0)}$ be equal to the identity matrix I, and for $t > 0$ we define $T^{(t)}$ as the matrix with entries $T^{(t)}(\mathbf{s}, \mathbf{s}') = Pr\big(\mathbf{X}(h + t) = \mathbf{s}' \,|\, \mathbf{X}(h) = \mathbf{s}\big)$. Note that $T^{(t)}$ is a stochastic matrix that does not depend on the time instant $h \geq 0$, its element $T^{(t)}(\mathbf{s}, \mathbf{s}')$ gives the probability for the Markov chain to be in state $\mathbf{s}'$ at time $t + h$ knowing that the Markov chain was in state $\mathbf{s}$ at time h, for all $h \geq 0$. Then the *Kolmogorov backward and forward equations* relate $T^{(t)}$ and Q by

$$\frac{dT^{(t)}}{dt} = QT^{(t)}, \quad \frac{dT^{(t)}}{dt} = T^{(t)}Q. \tag{2.6}$$

Let $\boldsymbol{p}^{(t)}$ be the row vector with entries $\boldsymbol{p}_{\mathbf{s}}^{(t)}$ for $\mathbf{s} \in \mathcal{S}$. We refer to the entries as *transient state probabilities*. The CME (see Equation (2.4)) is obtained from Equation (2.6) by multiplying both sides with $\boldsymbol{p}^{(0)}$. A general solution of the

CME is given by $\boldsymbol{p}^{(t)} = \boldsymbol{p}^{(0)} e^{Qt}$ and if Q is finite, from the definition of the matrix exponential

$$\boldsymbol{p}^{(t)} = \boldsymbol{p}^{(0)} e^{Qt} = \boldsymbol{p}^{(0)} \sum_{k=0}^{\infty} \frac{(Qt)^k}{k!}. \tag{2.7}$$

An analytic solution for the vector $\boldsymbol{p}^{(t)}$ can however only be derived for special cases, such as in the case of a birth-death structure. If the underlying graph of the CTMC is acyclic, a closed-form expression for $p_{\mathbf{s}}^{(t)}$ can be calculated using the recursive scheme of the ACE algorithm [77]. In general, finding the state probabilities as a symbolic function of t is not possible. If Q is finite and the number of nonzero entries is of manageable size, an approximate numerical solution can be computed. Adding up a sufficiently large number of terms of the infinite sum in Equation (2.7) is numerically unstable, as Q contains strictly positive and negative entries, leading to severe round-off errors [81]. Numerically stable methods are based on uniformization [46, 64] or approximations in the Krylov subspace [89]. Also numerical integration methods such as Runge-Kutta methods have been successfully used to compute $\boldsymbol{p}^{(t)}$. Several surveys and comparisons exist in literature [45, 101, 105]. For realistic systems, however, upper bounds on the state variables of the system are often not known and even if upper bounds are known, the size of the (truncated) state space is still too large for an efficient solution using standard approaches.

Most numerical solutions exploit the fact that the set $\{T^{(\tau)} \mid \tau \geq 0\}$, with $T^{(\tau)} = e^{Q\tau}$, is a *transition semi-group* and satisfies the *Chapman-Kolmogorov equations* [16] $T^{(\tau_1+\tau_2)} = T^{(\tau_1)} \cdot T^{(\tau_2)}$ for all $\tau_1, \tau_2 \geq 0$. Let $t_0, \ldots, t_r \in \mathbb{R}_{\geq 0}$ be such that $t_0 < \cdots < t_r$. Then,

$$\begin{aligned} \boldsymbol{p}^{(t_r)} &= \boldsymbol{p}^{(t_1)} \cdot T^{(t_2-t_1)} \cdot \ \ldots \ \cdot T^{(t_r-t_{r-1})} \\ &\quad\ \ddots \qquad\qquad\quad \vdots \\ &= \qquad\quad \boldsymbol{p}^{(t_{r-1})} \quad \cdot T^{(t_r-t_{r-1})}. \end{aligned} \tag{2.8}$$

This means that, for $t_0 = 0$ and $t_r = t$, we obtain $\boldsymbol{p}^{(t)}$ by iteratively applying Equation (2.6) for $t_1 - t_0, t_2 - t_1, \ldots, t_r - t_{r-1}$.

2.3 Propagation Models (PM)

We introduce propagation models, a general computational model for the modelling of processes that propagate a generic mass through a discrete state space.

We give both discrete and continuous time semantics to these models. In the discrete case, mass is propagated in steps, while in the continuous case states exchange mass between each other in a continuous flow.

Definition 2.0.2. *A* propagation model *is a tuple* $\langle \mathcal{S}, \mathcal{M}, \zeta, \pi \rangle$*, where:*

- *the discrete set* $\mathcal{S}$ *is the* state space *of the model,*
- *the ordered vector space* $\mathcal{M}$ *is the* mass space *of the model,*
- $\zeta \in [\mathcal{S} \to \mathcal{M}]$ *is the* initialization vector*, which assigns an initial mass value to each state,*
- $\pi : \mathcal{S} \times \mathcal{S} \times \mathcal{M} \to \mathcal{M}$ *is the* edge function*, which assigns a mass value to each pair of states, given the mass of the first state. For the propagation value from state* $\mathbf{s}$ *to state* $\mathbf{s}'$*, for a given mass value* μ *of* $\mathbf{s}$ *we use the notation* $\pi_{\mathbf{s}\to\mathbf{s}'}(\mu)$*.*

Example 2.3 (Predator-prey)**.** *The propagation model* N_1 *that represents the stochastic behaviour of a predator-prey model is defined over a state space* $\mathcal{S} = \mathbb{N}_0^2$*, where a state* $\mathbf{s}$ *of the system equals* (s_R, s_Y)*, and a mass space of probabilities:* $\mathcal{M} = [0, 1]$*. Furthermore,* $N_1 = \left\langle \mathbb{N}_0^2, [0, 1], \zeta, \pi \right\rangle$ *where:*

- $\zeta_{\mathbf{y}} = 1$*, for the initial state* $\mathbf{y} = (120, 30)$*, and* $\zeta_{\mathbf{s}} = 0$*, otherwise.*
- $\pi_{(s_R,s_Y)\to(s_R,s_Y+1)}(p) = p \cdot c_1 \cdot s_Y$*,*
- $\pi_{(s_R,s_Y)\to(s_R+1,s_Y-1)}(p) = p \cdot c_2 \cdot s_R \cdot s_Y$*,*
- $\pi_{(s_R,s_Y)\to(s_R-1,s_Y)}(p) = p \cdot c_3 \cdot s_R$*,*

As we will show later, there is an algorithmic construction of this PM N_1 *starting from its TCM, such that if* $p^{(t)}$ *is the solution of that TCM,* $\mathbf{p}^{(t)} = \boldsymbol{g}^{(t)}$*, where* $\boldsymbol{g}^{(t)}$ *is the propagation process of* N_1 *that we define in the next section.*

A propagation model is *mass-conservative* if it does not propagate mass on any self loop:

$$\pi_{\mathbf{s}\to\mathbf{s}}(\mu) = 0, \forall \mathbf{s} \in \mathcal{S}, \mu \in \mathcal{M}.$$

2.3.1 Discrete-time Propagation Process (DTPP)

For a given PM $N = \langle \mathcal{S}, \mathcal{M}, \zeta, \pi \rangle$ we define the *discrete-time propagation process* of N to be the function $\boldsymbol{f} : \mathbb{N}_0 \to [\mathcal{S} \to \mathcal{M}]$ defined on discrete time $\mathbb{N}_0$ and

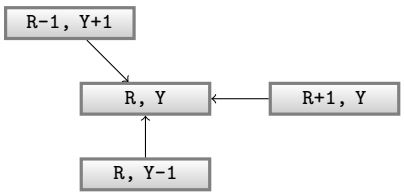


Figure 2.1: The states that propagate towards state (R,Y).

with values mass-vectors $\mathcal{S} \to \mathcal{M}$. The values $f^{(k)}_{\mathbf{s}'}$ are the elements of the mass-vector $\boldsymbol{f}^{(k)}$ and:

$$f^{(k)}_{\mathbf{s}'} = \begin{cases} \zeta_{\mathbf{s}'}, & \text{if } k = 0, \\ f^{(k-1)}_{\mathbf{s}'} + \sum\limits_{\mathbf{s}\in\mathcal{S}} \pi_{\mathbf{s}\to\mathbf{s}'}(f^{(k-1)}_{\mathbf{s}}) & \\ \qquad - \sum\limits_{\mathbf{s}''\in\mathcal{S}} \pi_{\mathbf{s}'\to\mathbf{s}''}(f^{(k-1)}_{\mathbf{s}'}), & \\ \qquad + \pi_{\mathbf{s}'\to\mathbf{s}'}(f^{(k-1)}_{\mathbf{s}'}), & \text{if } k > 0. \end{cases}$$

Example 2.4. *For the predator-prey system, Figure 2.1 shows how the predecessors of the state (R, Y), are propagating probability.*

2.3.2 Continuous-time Propagation Process (CTPP)

For a given PM $N = \langle \mathcal{S}, \mathcal{M}, \zeta, \pi \rangle$ we define the *continuous-time propagation process* of N to be the function $\boldsymbol{g} : \mathbb{R}_{\geq 0} \to [\mathcal{S} \to \mathcal{M}]$ that follows the following equations:

$$\begin{aligned} \frac{dg^{(t)}_{\mathbf{s}'}}{dt} &= \sum_{\mathbf{s}\in\mathcal{S}} \pi_{\mathbf{s}\to\mathbf{s}'}(g^{(t)}_{\mathbf{s}}) \\ &\quad - \sum_{\mathbf{s}''\in\mathcal{S}} \pi_{\mathbf{s}'\to\mathbf{s}''}(g^{(t)}_{\mathbf{s}'}) \\ &\quad + \pi_{\mathbf{s}'\to\mathbf{s}'}(g^{(t)}_{\mathbf{s}'}), \end{aligned} \tag{2.9}$$

with initial condition:

$$\boldsymbol{g}^{(0)} = \zeta$$

For an infinitesimal value dt, and for $t > 0$, the following holds:

$$g_{\mathbf{s}'}^{(t+dt)} = g_{\mathbf{s}'}^{(t)} + \sum_{\mathbf{s}\in\mathcal{S}} \pi_{\mathbf{s}\to\mathbf{s}'}\left(g_{\mathbf{s}}^{(t)}\right)\cdot dt - \sum_{\mathbf{s}''\in\mathcal{S}} \pi_{\mathbf{s}'\to\mathbf{s}''}\left(g_{\mathbf{s}'}^{(t)}\right)\cdot dt + \pi_{\mathbf{s}'\to\mathbf{s}'}\left(g_{\mathbf{s}'}^{(t)}\right)\cdot dt.$$

We observe that for conservative propagation models, that do not propagate mass on any self loops, the sum of all mass values in the system is always constant, both under discrete and the continuous-time semantics. E.g., in the discrete case we have the $\sum_{\mathbf{s}\in\mathcal{S}} f_{\mathbf{s}}^{(k)} = ct$, for all $k \geq 0$, and similarly for the continuous case. We need to allow self-loop propagation for the cases where the sum of mass values is not constant over time, as is the case, for example, when we need to propagate expectancies of variables, as will be exemplified in the second part of the thesis.

2.3.3 Linear Propagation Models

Linear propagation models are a special class of propagation models for which there exists a function $\pi' : \mathcal{S}\times\mathcal{S}\to\mathbb{R}$, such that for all $\mathbf{s},\mathbf{s}'\in\mathcal{S}$ and $\mu\in\mathcal{M}$:

$$\pi_{\mathbf{s}\to\mathbf{s}'}(\mu) = \mu\cdot\pi'_{\mathbf{s}\to\mathbf{s}'}.$$

Example 2.5. *The model given in Example 2.3 is indeed a linear propagation model, with:*

- $\pi'_{(s_R,s_Y)\to(s_R,s_Y+1)} = c_1\cdot s_Y$,
- $\pi'_{(s_R,s_Y)\to(s_R+1,s_Y-1)} = c_2\cdot s_R\cdot s_Y$,
- $\pi'_{(s_R,s_Y)\to(s_R-1,s_Y)} = c_3\cdot s_R$.

For linear propagation models the equations for discrete and continuous-time propagation processes, $\boldsymbol{f}^{(k)}$ and $\boldsymbol{g}^{(t)}$, can be written in a simple matrix form.

Let $N = \langle\mathcal{S},\mathcal{M},\zeta,\pi\rangle$ be a linear propagation model, and let P be the matrix with entries:

$$P(\mathbf{s},\mathbf{s}') = \begin{cases} \pi'_{\mathbf{s}\to\mathbf{s}'}, & \text{if } \mathbf{s}\neq\mathbf{s}', \\ 1+\pi'_{\mathbf{s}\to\mathbf{s}} - \sum_{\mathbf{s}''\in\mathcal{S},\mathbf{s}''\neq\mathbf{s}} \pi'_{\mathbf{s}\to\mathbf{s}''}, & \text{if } \mathbf{s}=\mathbf{s}'. \end{cases}$$

Lemma 2.1. *For discrete-time semantics we have that:*

$$\boldsymbol{f}^{(k+1)} = \boldsymbol{f}^{(k)} \cdot P$$

.

Proof. We compute the value of the vector $\boldsymbol{f}^{(k)} \cdot P$ for each $\mathbf{s}' \in \mathcal{S}$:

$$\begin{aligned}
(\boldsymbol{f}^{(k)} \cdot P)_{\mathbf{s}'} &= \sum_{\mathbf{s}\in\mathcal{S}} f^{(k)}_{\mathbf{s}} \cdot P(\mathbf{s},\mathbf{s}') \\
&= \sum_{\mathbf{s}\in\mathcal{S},\mathbf{s}\neq\mathbf{s}'} f^{(k)}_{\mathbf{s}} \cdot \pi'_{\mathbf{s}\to\mathbf{s}'} + f^{(k)}_{\mathbf{s}'} \cdot \big(1 + \pi'_{\mathbf{s}'\to\mathbf{s}'} - \sum_{\mathbf{s}\in\mathcal{S},\mathbf{s}\neq\mathbf{s}'} \pi'_{\mathbf{s}'\to\mathbf{s}}\big) \\
&= f^{(k)}_{\mathbf{s}'} + \sum_{\mathbf{s}\in\mathcal{S}} f^{(k)}_{\mathbf{s}} \cdot \pi'_{\mathbf{s}\to\mathbf{s}'} + f^{(k)}_{\mathbf{s}'} \cdot \pi'_{\mathbf{s}'\to\mathbf{s}'} - \sum_{\mathbf{s}\in\mathcal{S}} f^{(k)}_{\mathbf{s}'} \cdot \pi'_{\mathbf{s}'\to\mathbf{s}} \\
&= f^{(k)}_{\mathbf{s}'} + \sum_{\mathbf{s}\in\mathcal{S}} \pi_{\mathbf{s}\to\mathbf{s}'}(f^{(k)}_{\mathbf{s}}) + \pi_{\mathbf{s}'\to\mathbf{s}'}(f^{(k)}_{\mathbf{s}'}) - \sum_{\mathbf{s}\in\mathcal{S}} \pi_{\mathbf{s}'\to\mathbf{s}}(f^{(k)}_{\mathbf{s}'}) \\
&= f^{(k+1)}_{\mathbf{s}'}.
\end{aligned}$$

□

From $\boldsymbol{f}^{(k+1)} = \boldsymbol{f}^{(k)} \cdot P$ it follows that $\boldsymbol{f}^{(k)} = \boldsymbol{f}^{(0)} \cdot P^k$, and in Chapter 3 we present ways of computing this matrix-vector multiplication for very large systems. Also, from Lemma 2.1 it follows that the matrix P completely encodes the function π of N.

Next, for the continuous-time case, we define the matrix Q as follows:

$$Q(\mathbf{s},\mathbf{s}') = \begin{cases} \pi'_{\mathbf{s}\to\mathbf{s}'}, & \text{if } \mathbf{s} \neq \mathbf{s}', \\ \pi_{\mathbf{s}\to\mathbf{s}} - \sum_{\mathbf{s}''\in\mathcal{S},\mathbf{s}''\neq\mathbf{s}'} \pi'_{\mathbf{s}''\to\mathbf{s}}, & \text{if } \mathbf{s} = \mathbf{s}'. \end{cases}$$

Note, we have $P = I + Q$.

Lemma 2.2. *For continuous-time semantics, $\boldsymbol{g}$ is the solution of the differential equation:*

$$\frac{d\boldsymbol{g}^{(t)}}{dt} = \boldsymbol{g}^{(t)} \cdot Q, \text{ with initial condition: } \boldsymbol{g}^{(0)} = \zeta. \tag{2.10}$$

We call this equation the propagation master equation (PME), in analogy with the chemical master equation presented in Section 2.2.

Proof. We compute the value of the vector $\boldsymbol{g}^{(t)} \cdot Q$ for each $\mathbf{s}' \in \mathcal{S}$:

$$\begin{aligned}
(\boldsymbol{g}^{(t)} \cdot Q)_{\mathbf{s}'} &= \sum_{\mathbf{s}\in\mathcal{S}} g_{\mathbf{s}}^{(t)} \cdot Q(\mathbf{s}, \mathbf{s}') \\
&= g_{\mathbf{s}'}^{(t)} \cdot \pi'_{\mathbf{s}'\to\mathbf{s}} + \sum_{\mathbf{s}\in\mathcal{S},\mathbf{s}\neq\mathbf{s}'} g_{\mathbf{s}}^{(t)} \cdot \pi'_{\mathbf{s}\to\mathbf{s}'} - \sum_{\mathbf{s}\in\mathcal{S},\mathbf{s}''\neq\mathbf{s}'} g_{\mathbf{s}'}^{(t)} \cdot \pi'_{\mathbf{s}'\to\mathbf{s}} \\
&= g_{\mathbf{s}'}^{(t)} \cdot \pi'_{\mathbf{s}'\to\mathbf{s}} + \sum_{\mathbf{s}\in\mathcal{S}} g_{\mathbf{s}}^{(t)} \cdot \pi'_{\mathbf{s}\to\mathbf{s}'} - \sum_{\mathbf{s}\in\mathcal{S}} g_{\mathbf{s}'}^{(t)} \cdot \pi'_{\mathbf{s}'\to\mathbf{s}} \\
&= \pi_{\mathbf{s}'\to\mathbf{s}'}(g_{\mathbf{s}'}^{(t)}) + \sum_{\mathbf{s}\in\mathcal{S}} \pi_{\mathbf{s}\to\mathbf{s}'}(g_{\mathbf{s}}^{(t)}) - \sum_{\mathbf{s}\in\mathcal{S}} \pi_{\mathbf{s}'\to\mathbf{s}}(g_{\mathbf{s}'}^{(t)}) \\
&= \frac{dg_{\mathbf{s}'}^{(t)}}{dt}.
\end{aligned}$$

□

The Equation 2.9 has the solution:

$$\mathbf{g}^{(t)} = \boldsymbol{g}^{(0)} \cdot e^{Q\cdot t} = g^{(0)} \cdot \sum_{k=0}^{\infty} \frac{(Qt)^k}{k!}. \tag{2.11}$$

As in the discrete-time case, from Lemma 2.2 it follows that the matrix Q completely encodes the function π of N. Therefore, given either matrix Q or P, and the initial mass vector ζ we can construct the associated PM.

In Chapter 4, we present several algorithms for computing approximative solutions of Equation (2.11). Throughout this entire work we will use the same notations:

M	a transition class model
$X(t)$	a continuous-time Markov chain (CTMC)
$X(k)$	a discrete-time Markov chain (DTMC)
$p^{(t)}$	a transient solution of $X(t)$
$p^{(k)}$	the solution of $X(k)$
N	a propagation model
$g^{(t)}$	the continuous-time propagation process (CTPP) of N
$f^{(k)}$	the discrete-time propagation process (CTPP) of N

2.3.4 Relation with Transition Class Models

Propagation models are a generalization of TCMs. They make the propagation mass explicit, and thus can express more general behaviours. Transition class

models have the advantage of giving a very structural description of Markov chains, but, as we will see later, encounter limitation when we want to define stochastic processes over an aggregated space, because aggregation is breaking the transition classes.

We will now show how the probabilities $p_{\mathbf{s}}^{(t)}$, defined for a TCM M can be expressed through the continuous-time propagation process $\boldsymbol{g}$ of a propagation model N.

For a given TCM M, we derive a PM N, such that:

$$p_{\mathbf{s}}^{(t)} = g_{\mathbf{s}}^{(t)}, \forall t \geq 0, \mathbf{s} \in \mathcal{S}.$$

Definition 2.2.1 (PM of a TCM)**.** *Given a TCM $M = \langle \mathcal{S}, \mathbf{y}, \{\mathcal{C}_1, \ldots, \mathcal{C}_m\}\rangle$, we construct the PM $N = \langle \mathcal{S}, \mathcal{M}, \zeta, \pi\rangle$ as follows:*

- *the mass space $\mathcal{M} = [0, 1]$,*
- *the initial mass vector ζ is:*

$$\zeta_{\mathbf{s}} = \begin{cases} 1, & \text{if } \mathbf{s} = \mathbf{y}, \\ 0, & \text{otherwise,} \end{cases}$$

- *the edge function π is:*

$$\pi_{\mathbf{s}\to\mathbf{s}'}(\mu) = \begin{cases} \mu \cdot \alpha_j(\mathbf{s}), & \text{if } \exists j.u_j(\mathbf{s}) = \mathbf{s}', \\ 0, & \text{otherwise.} \end{cases}$$

Recall that from the injectivity property of the update functions, there exists at most one j such that $u_j(\mathbf{s}) = \mathbf{s}'$, and thus the PM above is well defined.

Lemma 2.3. *If N is the derived PM of M then $\boldsymbol{g}$, the continuous-time propagation process of N, equals the probability vector associated to M:*

$$p_{\mathbf{s}}^{(t)} = g_{\mathbf{s}}^{(t)}, \forall t \geq 0, \forall \mathbf{s} \in \mathcal{S}.$$

Proof. The two vectors: $\mathbf{p}^{(t)}$ and $\mathbf{g}^{(t)}$ obey the same differential equation, and have the same initial condition. □

Note that the propagation model that we have just constructed is both linear and conservative.

Part I

Propagation Algorithms

Introduction

In this part we present various algorithms that solve both discrete-time and continuous-time propagation problems. Computing exact solutions for the continuous time case is almost never possible, so each of the algorithms that we present has its own way of approximating the exact solution. For discrete-time problems, computing an exact solution is possible but can be computationally expensive, so even in the discrete-time case an approximation is usually desired. This is especially true if the discrete-time algorithm is called by a method that approximates a continuous-time solution. As we have said, an exact solution of the continuous-time problem is not possible, and therefore, computing an exact solution of the discrete-time problem is not needed in this case.

We start by presenting algorithms for the discrete-time case, where we make use of on-the-fly construction of the state space and of threshold abstraction in order to make our algorithms efficient.

Then, we introduce methods for the continuous-time case. Sliding window is presented for the specific case of transition class models, while uniformization and Runge-Kutta integration methods are presented for general propagation models. While the Runge-Kutta method can be applied in a straight-forward way to evaluate propagation processes, uniformization and adaptive uniformization are generalized from Markov chains to propagation processes.

Chapter 3

Discrete Time Propagation

3.1 Introduction

In this chapter we show how to evaluate the discrete-time propagation process $\boldsymbol{f}^{(k)}$ of a given propagation model N. We first present a straight-forward version of the algorithm, and then we show two techniques that mitigate the performance problems of numerical solution algorithms for propagation models that have a very large or infinite state space. The first technique that we present is the on-the-fly construction of the state space, which adds new states to the solution as they are discovered (their mass becomes strictly positive). And the second technique involves a problem relaxation by which an approximative solution, within a given error tolerance, is required.

3.2 Straight-forward Solution

We first formulate the exact DTPP problem and give a straight-forward solution that only works for finite spaces.

Problem 3.1 (DTPP problem)**.** *Given:*

– *a propagation model N,*

– *an integer $k \geq 0$,*

compute the discrete-time mass vector $\boldsymbol{f}^{(k)}$.

Field	Type	Description
$x.\mathbf{s}$	array of integers	state $\mathbf{s} \in \mathcal{S}$
$x.\mu$	real	mass $f_{\mathbf{s}}^{(k)}$
$x.acc$	real	variable in which all propagated mass is added

Table 3.1: Structure of node x of state $\mathbf{s}$.

Algorithm 3.1 main_v1

Input: finite propagation model $N = \langle \mathcal{S}, \mathcal{M}, \zeta, \pi \rangle$, time horizon k;
Output: mass vector $\boldsymbol{f}^{(k)}$ will be stored in $\mathbf{S}.\mu$ at the end of the algorithm;
Variables: space structure $\mathbf{S}$.

```
1:  // construct a structure with nodes for each state in S.
2:  for all s ∈ S do
3:     x.s ← s;  // initialize the state field
4:     x.μ ← ζ_s;  // initialize the mass field
5:     x.acc ← 0;  // initialize the accumulate field
6:     S ← S ∪ {x};
7:  for k' ← 1...k do
8:     propagate(S, N);
9:     collect(S);
10: return S.μ.
```

In this first version of the algorithm $\mathbf{S}$ is a static data structure for the state space $\mathcal{S}$, where we associate with each node $x \in \mathbf{S}$ three fields, as listed in Table 3.1. The field $\mathbf{s}$ represents the state $\mathbf{s} \in \mathcal{S}$ that is associated to the node, the field μ holds the value $f_{\mathbf{s}}^{(k)}$, while the field acc is used to accumulate the incoming propagation mass for the next value of the field μ. In the following, we refer to the fields associated to x as $x.\mathbf{s}$, $x.\mu$ and $x.acc$. Note that the fields $x.\mu$ and $x.acc$ of each node $x \in \mathbf{S}$ are initialized with zero.

Algorithm 3.1 gives the pseudocode of the main loop of the algorithm. In lines 1-6, we construct the structure $\mathbf{S}$ with nodes for the entire state space $\mathcal{S}$ of the propagation model N. The structure will therefore include nodes even for states that are not needed for the solution at time k, because some states are not reachable in k steps from the initial state of the model. This limits the use of this first algorithm to propagation models with a finite state space. The mass value of each node is initialized on line 4 according to the initialization function ζ of N.

The computation of the solution at time step k is done iteratively by computing each solution at times $1 \dots k$, in the loop at lines 7-9, and at the end of each iteration k', the value $f_{\mathbf{s}}^{(k')}$ is stored in the field $x.\mu$ of the node x

Algorithm 3.2 propagate

Input: space structure $\mathbf{S}$, propagation model $N = \langle \mathcal{S}, \mathcal{M}, \zeta, \pi \rangle$.

1: **for all** $x \in \mathbf{S}$ **do**
2: **for all** $\mathbf{s}'$ such that $\pi_{x.\mathbf{s}\to\mathbf{s}'}(x.\mu) > 0$ **do**
3: $x' = find(\mathbf{s}', \mathbf{S})$;
4: $x'.acc \leftarrow x'.acc + \pi_{x.\mathbf{s}\to\mathbf{s}'}(x.\mu)$;
5: $x.acc \leftarrow x.acc - \pi_{x.\mathbf{s}\to\mathbf{s}'}(x.\mu)$;

Algorithm 3.3 collect

Input: space structure $\mathbf{S}$.

1: **for all** $x \in \mathbf{S}$ **do**
2: $x.\mu \leftarrow x.\mu + x.acc$;
3: $x.acc \leftarrow 0$;

that corresponds to the state $\mathbf{s}$ (e.g. $x.\mathbf{s} = \mathbf{s}$). Each computation step has two phases, the propagate phase and the collect phase that push and pull the propagated mass through the state space. The *propagate* method (presented in Algorithm 3.2) iterates over all nodes $x \in \mathbf{S}$. For all successor states $\mathbf{s}'$ of $x.\mathbf{s}$ (i.e. $\pi_{x.\mathbf{s}\to\mathbf{s}'}(f_{\mathbf{s}}^{(k')}) > 0$), we move mass from x to the node corresponding to $\mathbf{s}'$. During the computation of $\boldsymbol{f}^{(k'+1)}$, the field $\mathbf{s}'.\mu$ represents the mass value at step k': $f_{\mathbf{s}'}^{(k')}$, and it cannot be altered before being used at its own turn by the loop at line 1 of the propagate method. This is why we need to use the field *acc* to accumulate the mass that is being propagated by the predecessors of $\mathbf{s}'$. The final result $\boldsymbol{f}^{(k'+1)}$ is calculated in method *collect* where all incoming probability mass of a state $\mathbf{s}$ is added up (see line 2 of Algorithm 3.3).

This first version of the algorithm is exact, but very expensive, as it needs to construct a structure for the entire state space $\mathcal{S}$. For the same reason, this method is limited to finite propagation models. In the next two sections we show how to construct the state space on-the-fly and then how to approximate the solution $\boldsymbol{f}^{(k)}$ by means of abstraction.

3.3 On-the-fly State Space

A first improvement that we can bring to our algorithm is to construct the state space on-the-fly rather than a-priori. In this new version of the algorithm, the *propagate* and the *collect* methods are moving mass through the system in the same time as they discover new states.

In this new version, $\mathbf{S}$ is a dynamic data structure for the state space $\mathcal{S}$, where states are added as they are discovered by the propagation method. Algorithm 3.4 gives the pseudocode for the main loop of the on-the-fly algorithm.

Algorithm 3.4 main_v2

Input: propagation model $N = \langle \mathcal{S}, \mathcal{M}, \zeta, \pi \rangle$, time horizon k,
Output: mass-vector $\boldsymbol{f}^{(k)}$,
Variables: space structure $\mathbf{S}$.

```
1:  // initialize S with the support of ζ.
2:  for all s ∈ S with ζ_s > 0 do
3:     x.s ← s;  // initialize the state field
4:     x.μ ← ζ_s;  // initialize the mass field
5:     x.acc ← 0;  // initialize the accumulate field
6:     S ← S ∪ {x};
7:  for k' ← 1 ... k do
8:     propagate_and_add(S, N);
9:     collect(S);
10: return S.μ.
```

Algorithm 3.5 propagate_and_add

Input: space structure $\mathbf{S}$, propagation model $N = \langle \mathcal{S}, \mathcal{M}, \zeta, \pi \rangle$.

```
1: for all x ∈ S do
2:    for all s' such that π_{x.s→s'}(x.μ) > 0 do
3:       x' ← find(s', S);
4:       if x' = null then
5:          // x' is not yet in the space.
6:          x' ← init_node(s');
7:          S ← S ∪ {x'};
8:       x'.acc ← x'.acc + π_{x.s→s'}(x.μ);
9:       x.acc ← x.acc − π_{x.s→s'}(x.μ);
```

The differences from the first version (Algorithm 3.1) are that the state space is initialized with the support of the function ζ and the call to the function *propagate* is replaced with a call to the function *propagate_and_add*. The *propagate_and_add* method (see Algorithm 3.5) is checking if state $\mathbf{s}'$ is already part of the current space structure, and if this is not the case a new node x' is created and then added to $\mathbf{S}$ (lines 4-7).

We define $\mathcal{S}_{k'}$ to be the set of states of $\mathcal{S}$ that are *reachable* in k' steps:

$$\mathcal{S}_{k'} = \begin{cases} \{\mathbf{s}' \mid \zeta_{\mathbf{s}'} > 0\}, & \text{if } k' = 0, \\ \left\{\mathbf{s}' \mid \mathbf{s} \in \mathcal{S}_{k'-1}, \pi_{\mathbf{s}\to\mathbf{s}'}(f_{\mathbf{s}}^{(k'-1)}) > 0\right\} & \text{if } k' > 0. \end{cases}$$

According to this definition, at the k' iteration, the function *propagate_and_add* is adding nodes for all states in $\mathcal{S}_{k'+1} \setminus \mathcal{S}_{k'}$ to the current space structure.

For now, the structure $\mathbf{S}$ is dynamic in the sense that states are dynamically added to it. It is only in the next version of the algorithm, presented in Section 3.4, that states that are no longer needed by the algorithm are also dynamically removed from the state space.

In this section we have shown how to integrate the construction of the state space in the propagation algorithm by constructing the state space on-the-fly. This technique simplifies our algorithm, and it can also be applied to propagation models with an infinite state space, because even for such model the set of states reachable in k steps, $\mathcal{S}_k$, is finite. This method offers an improvement in performance as the construction of the entire state space can be computationally expensive. In the next section, in order to obtain an even greater performance, we propose a problem relaxation as a way to compromise between the accuracy of the algorithm and the speed of its execution.

3.4 Problem Relaxation

For very large systems, the problem of computing the mass-vector, $\boldsymbol{f}^{(k)}$ exactly is intractable. In this section we propose a relaxation of this problem, relaxation by which we will be able to obtain major improvements in performance.

The relaxed DTPP problem asks for the computation of an under-approximation $\boldsymbol{f}^{\text{-}}$ such that all elements of the mass-vector $\boldsymbol{f}^{\text{-}}$ are greater then a threshold δ. The relaxed problem also asks for the error ϵ to be computed.

Problem 3.2 (Relaxed DTPP problem)**.** – *a propagation model N, with mass space $\mathcal{M}$,*

– *an integer $k \geq 0$, and*

– *a cutting threshold $\delta \in \mathcal{M}$,*

find a vector $\boldsymbol{f}^{\text{-}(k)}$ and a minimal value ϵ such that:

– $f^{\text{-}(k)}_{\mathbf{s}} \leq f^{(k)}_{\mathbf{s}}$, *for all* $\mathbf{s} \in \mathcal{S}$, *and*

– $f^{\text{-}(k)}_{\mathbf{s}} \geq \delta$ *or* $f^{\text{-}(k)}_{\mathbf{s}} = 0$,

– $\sum_{\mathbf{s}\in\mathcal{S}} f^{\text{-}(k)}_{\mathbf{s}} = 1 - \epsilon$.

3.4.1 Threshold Abstraction

The threshold abstraction is replacing the vectors $\boldsymbol{f}^{(0)}, \boldsymbol{f}^{(1)}, \ldots$ with the approximations $\boldsymbol{f}^{\text{-}(0)}, \boldsymbol{f}^{\text{-}(1)}, \ldots$ while keeping the number of non-zero entries small. Assume that δ is a small positive threshold. We first define $\boldsymbol{f}^{\text{-}(0)}$ as the vector that is derived from the initial mass vector ζ by setting all entries that are

Algorithm 3.6 main_v3

Input: propagation model $N = \langle \mathcal{S}, \mathcal{M}, \zeta, \pi \rangle$, time horizon k, threshold $\delta \in \mathcal{M}$
Output: approximation mass-vector $\boldsymbol{f}^{\bar{}(k)}$, error ϵ
Variables: space structure $\mathbf{S}$.

```
1:  // initialize S with the support of ζ.
2:  for all s ∈ S with ζ_s > δ do
3:     x.s ← s;  // initialize the state field
4:     x.μ ← ζ_s;  // initialize the mass field
5:     x.acc ← 0;  // initialize the accumulate field
6:     S ← S ∪ {x};
7:  for k' ← 1 ... k do
8:     propagate_and_add(S, N);
9:     collect_and_remove(S, δ);
10: ε ← 1 − sum(S.μ);
11: return S.μ, ε.
```

Algorithm 3.7 collect_and_remove

Input: space structure $\mathbf{S}$, threshold δ.

```
1: for all x ∈ S do
2:    x.μ ← x.μ + x.acc;
3:    x.acc ← 0;
4:    if x.μ ≤ δ then
5:       S ← S \ {x};
```

smaller or equal to δ to zero. We compute the entries of the vector $\boldsymbol{f}^{\bar{}(k+1)}$ from $\boldsymbol{f}^{\bar{}(k)}$ using[1]

$$\begin{aligned} f^{\bar{}(k+1)}_{\mathbf{s}} = f^{\bar{}(k)}_{\mathbf{s}} &+ \sum_{\substack{\mathbf{s}': \\ f^{\bar{}(k)}_{\mathbf{s}'} > \delta}} \pi_{\mathbf{s}' \to \mathbf{s}}(f^{\bar{}(k)}_{\mathbf{s}'}) \\ &- \sum_{\mathbf{s}'} \pi_{\mathbf{s} \to \mathbf{s}'}(f^{\bar{}(k)}_{\mathbf{s}}) \\ &+ \pi_{\mathbf{s} \to \mathbf{s}}(f^{\bar{}(k)}_{\mathbf{s}}). \end{aligned} \tag{3.1}$$

Note that if $\delta = 0$ the above approximation is exact. For our experimental results we vary δ between 10^{-11} and 10^{-15}. For DTPPs derived from a chemical master equation, this approximation is especially accurate and we obtain considerable computational savings. The reason is that in each step the probability mass concentrates on a relatively small number of "significant" states while the probability of the remaining states decreases the further away a state is from regions where lots of probability mass is located. In particular, if the state space is infinite, then this decrease typically has an exponential order of magnitude.

[1] In the sequel, we add the $\bar{}$-symbol whenever we make use of the threshold abstraction, i.e. whenever we neglect entries in a vector that are below a certain threshold.

In Algorithm 3.6 we show the main loop of the modified algorithm. The main differences with respect to the previous version is that in the initialization phase we disregard states that have an initial mass lower than δ (line 2) and that we change the call to the *collect* method with a call to the *collect_and_remove* method which besides collecting the propagated values also removes nodes with a mass lower than δ (lines 4-5 of Algorithm 3.7).

3.4.2 Error Tolerance

We make a last improvement to our algorithm, by which the cutting threshold δ is adjusted dynamically such that a given error tolerance *tol* is achieved.

Problem 3.3 (Error Tolerance DTPP Problem)**.** *Given:*

- *a propagation model N, with mass space $\mathcal{M}$, where $\mathcal{M}$ is an ordered vector space,*
- *an integer $k \geq 0$, and*
- *error tolerance $tol \in \mathcal{M}$,*

find a vector $\boldsymbol{f}^{\text{-}(k)}$ such that:

- $f^{\text{-}(k)}_{\mathbf{s}} \leq f^{(k)}_{\mathbf{s}}$, *for all* $\mathbf{s} \in \mathcal{S}$, *and*
- $\sum_{\mathbf{s}\in\mathcal{S}} f^{\text{-}(k)}_{\mathbf{s}} \geq 1 - tol$.

In order to solve this problem, we propose to compute after each iteration of the main loop the error that has been already accumulated, and in case the error is too large, to decrease the cutting threshold and to recompute until the error is within the desired error tolerance.

The main loop of the algorithm is presented in Algorithm 3.8, where the loop at lines 10-19 has the role of checking if the current error is within the tolerated error. More exactly, at the k'-th iteration, the error should not be larger than $tol \cdot \frac{k'}{k}$, or else a recomputation is needed. Note that this approach requires for a backup $\mathbf{S}_{bkp}$ of the space structure to be made before the call of the propagate/collect methods, in case a recomputation will be needed (lines 9 and 15).

Some solutions of the continuous time problem for which we give experimental results in the next chapter, rely on algorithms for the discrete-time case. We have found that, in the case of biological systems, Algorithm 3.6 reports

Algorithm 3.8 main_v4

Input: propagation model $N = \langle \mathcal{S}, \mathcal{M}, \zeta, \pi \rangle$, time horizon k, tolerance tol,
Output: approximation mass-vector $\boldsymbol{f}^{-(k)}$,
Variables: space structure $\mathbf{S}$.

1: // initialize $\mathbf{S}$ with the support of ζ.
2: **for all** $\mathbf{s} \in \mathcal{S}$ with $\zeta_{\mathbf{s}} > \delta$ **do**
3: $x.\mathbf{s} \leftarrow \mathbf{s}$; // initialize the state field
4: $x.\mu \leftarrow \zeta_{\mathbf{s}}$; // initialize the mass field
5: $x.acc \leftarrow 0$; // initialize the accumulate field
6: $\mathbf{S} \leftarrow \mathbf{S} \cup \{x\}$;
7: $\delta \leftarrow 1{\times}10^{-10}$;
8: **for** $k' \leftarrow 1 \dots k$ **do**
9: $\mathbf{S}_{bkp} \leftarrow \mathbf{S}$;
10: **repeat**
11: $propagate\_and\_add(\mathbf{S}, N)$;
12: $collect\_and\_remove(\mathbf{S}, \delta)$;
13: **if** $(1 - sum(\mathbf{S}.\mu)) > tol \cdot \frac{k'}{k}$ **then**
14: $ok = false$;
15: $\mathbf{S} \leftarrow \mathbf{S}_{bkp}$;
16: $\delta \leftarrow \delta/10$;
17: **else**
18: $ok = true$;
19: **until** $ok = true$;
20: **return** $\mathbf{S}.\mu$.

small errors for cutting thresholds in the order of $1{\times}10^{-14}$ and as it does not have the overhead of making a backup of the current solution, we prefer it to Algorithm 3.8 that we have just presented. Therefore, from now one, all calls to discrete-time solutions refer to Algorithm 3.6.

Chapter 4

Continuous Time Propagation

4.1 Introduction

In this chapter we give different strategies to approximate the *CTPP problem.*

Problem 4.1 (CTPP problem)**.** *Given:*

– *a propagation model N,*

– *a real number $t \geq 0$,*

compute continuous-time mass vector $\boldsymbol{g}^{(t)}$.

We discuss two methods of solving the CTPP problem: reducing the state space of the system and reducing the problem to a DTPP problem. These two methods are orthogonal. Reducing the state space of the system can be seen in parallel to the threshold abstraction proposed in Chapter 3.

First, we introduce the sliding window method, which computes an approximate solution of the CME by performing a sequence of local analysis steps. In each step, only a manageable subset of states is considered, representing a "window" into the state space. In subsequent steps, the window follows the direction in which the probability mass moves, until the time period of interest has elapsed. We construct the window based on a deterministic approximation of the future behavior of the system by estimating upper and lower bounds on the populations of the chemical species.

Second, we deal with reducing general CTPP to DTPP, by using classical numerical solution algorithms, namely the uniformization method, the Runge-Kutta integration method.

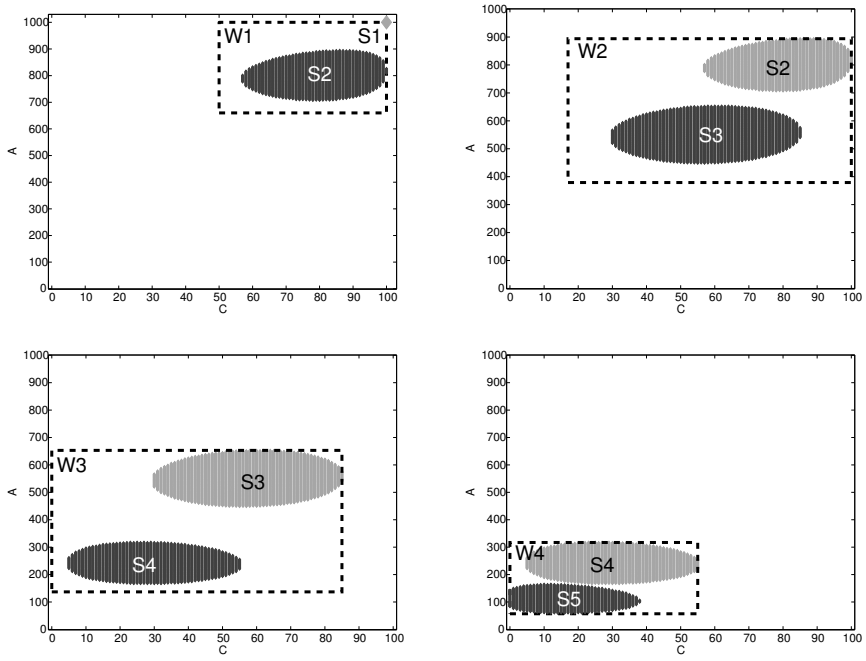


Figure 4.1: Sliding window method. In each iteration step, the window W_r captures the set S_r of states where the significant part of the probability mass is located initially (light gray), the set S_{r+1} of states that are reached after a time step (dark gray), and the states that are visited in between.

4.2 Sliding Window

We present the sliding window method for transition class models that represent biochemical reaction networks. As an alternative to a global analysis of the state space, we propose the *sliding window method*, which comprises a sequence of analyses local to the significant parts of the state space. In each step of the sequence, we dynamically choose a time interval and calculate an approximate numerical solution for a manageable subset of the reachable states.

In order to identify those states that are relevant during a certain time period, for each variable of the system, we estimate an upper and a lower bound on its value. This yields the boundaries of a "window" in which most of the probability mass remains during the time interval of interest. As illustrated in Figure 4.1, the window "slides" through the state space when the system is analysed in a stepwise fashion. In each step, the initial conditions are given by a vector of probabilities (whose support is illustrated in light gray), and a matrix is constructed to describe the part of the Markov process where the window (illustrated by the dashed rectangular) is currently located. Then the

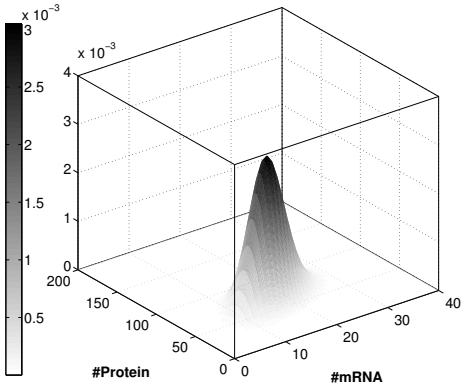


Figure 4.2: Probability distribution of the gene expression network at $t = 1000$.

corresponding ODE is solved using a standard numerical algorithm, and the next vector (illustrated in dark gray) is obtained.

Our approach is based on the observation that a Markov chain describing a certain real-world system often has the following property. After starting with probability 1 in the initial state $\mathbf{y}$, the probability mass does not distribute uniformly in $\mathcal{S}$, such as, for instance, in the case of a random walk. Instead, at each point in time, most of the probability mass distributes among a finite, relatively small number of states. This set of states changes as time progresses, but it never exceeds a certain size. Often, the states with "significant" probability are located at the same part of the state space, as illustrated in Figure 4.2.

Formally, for $\epsilon > 0$ and for an interval $[t, t+h)$, we define the size $m(\epsilon, t, h)$ of the set of significant states as the smallest number for which there exists $W \subseteq \mathcal{S}$, $|W| = m(\epsilon, t, h)$ such that

$$P\left(\mathbf{X}(t') \in W, t' \in [t, t+h)\right) \geq 1 - \epsilon. \tag{4.1}$$

The value $m(\epsilon, t, h)$ indicates how strongly the probability mass spreads out on $\mathcal{S}$ during $[t, t+h)$. Consider, for instance, a random walk on the non-negative integer lattice in the plane that starts in $(0, 0)$ [85]. Between each pair of neighbour states there is a transition with rate 1. For $h > 0$, the value $m(\epsilon, t, h)$ approaches infinity as $t \to \infty$. As opposed to the random walk example, in many systems $m(\epsilon, t, h)$ is a manageable number of states, even if ϵ is small and t is large (or tends to infinity). Consider, for instance, the gene expression example (see Example 2.1) and assume that $h = 500$, $\epsilon = 10^{-6}$. For each interval $[t, t+h) \subset [0, \infty)$, $m(\epsilon, t, h)$ does not exceed 20000 states. The

sliding window algorithm works well if $m(\epsilon, t_{r-1}, h_r)$ is a manageable number of states for all r. Note that, in particular, cellular systems usually follow a small number of trends, that is, the quantitative outcomes of a biological experiments can usually be classified within a small number of different categories. Thus, this assumption is acceptable for TCMs of biological systems.

4.2.1 Sliding Window Abstraction

Let $M = \langle \mathcal{S}, \mathbf{y}, \{\mathcal{C}_1, \ldots, \mathcal{C}_m\}\rangle$ be a TCM and let $\boldsymbol{p}^{(t)}$ be the probability distribution vector of the associated CTMC. We propose an abstraction technique for the computation of $\boldsymbol{p}^{(t)}$ that proceeds in an iterative fashion. We divide the time interval $[0, t)$ into r' intervals as in Equation (2.8) and approximate the vectors $\mathbf{p}^{(t_1)}, \ldots, \mathbf{p}^{(t_{r'})}$ by $\hat{\mathbf{p}}^{(t_1)}, \ldots, \hat{\mathbf{p}}^{(t_{r'})}$. In order to obtain these approximations, we consider a sequence of r' abstractions of the Markov chain under study. Let $r \in \{1, \ldots, r'\}$. In the r-th step, we construct, on-the-fly, a finite Markov chain for the system behaviour during the interval $[t_{r-1}, t_r)$ from the transition class description. The state space of the r-th abstract model is the set W_r of states where most of the probability mass is located during $[t_{r-1}, t_r)$. We refer to this set as a *window*. The remaining states are collapsed into a single absorbing state s_f, i.e., a state that cannot be left.

For $r \in \{1, \ldots, r'\}$, let W_r be such that

$$P\left(\mathbf{X}(h) \in W_r, h \in [t_{r-1}, t_r)\right) \geq 1 - \epsilon_r \tag{4.2}$$

where $\epsilon_r > 0$ is the approximation error of the r-th step. Note that Equation (4.2) implies that $W_r \cap W_{r+1} \neq \emptyset$, because the intersection of two successive windows must contain those states that have a high probability at time t_r. It is far too costly to construct the *smallest* set with this property. Instead, we propose a cheap construction of a set W_r with a hyper-rectangular shape. We will outline the construction in Section 4.2.2. The abstract Markov chain of the r-th step has the finite state space $W_r \cup \{s_f\}$, where s_f represents all states $\mathbf{s} \in \mathcal{S} \setminus W_r$. The transitions of the abstract model are given by the transition classes of the original model except that all transitions of states at the boundary lead to s_f. Formally, for each class $\mathcal{C} = (G, u, \alpha)$ of the infinite-state Markov chain $(\mathbf{X}(t))_{t\geq 0}$, we define $\mathcal{C}' = (G', u', \alpha')$ such that $G' = G \cap W_r$,

$$u'(\mathbf{s}) = \begin{cases} u(\mathbf{s}) & \text{if } u(\mathbf{s}) \in W_r, \\ s_f & \text{otherwise,} \end{cases}.$$

and $\alpha'(\mathbf{s}) = \alpha(\mathbf{s})$ for all $\mathbf{s} \in G'$. Thus, we consider an (extended) subgraph of the one underlying $\mathbf{X}(t)$, with vertexes set W_r, and all edges leading from W_r to $\mathcal{S} \setminus W_r$ redirected to the extension s_f. Note that no transitions are possible from s_f. We will see that s_f can be used to calculate the approximation error as it captures the probability mass that leaves W_r.

4.2.2 Window Construction

Let us now focus on the construction of the set W_r used in the sliding window abstraction (see Equation (4.2)). Recall that this requires the prediction of the size and location of the probability mass during $[t_{r-1}, t_r)$ of length $h_r = t_r - t_{r-1}$. For arbitrary transition class models, a cheap prediction of the future behaviour of the process is not possible as the transition classes may describe any kind of "unsystematic" behaviour. However, many systems have certain linearity properties that allow for an efficient approximation of the future behaviour of the process. Consider a transition class $\mathcal{C}_j = (G_j, u_j, \alpha_j)$, and assume that the successor $u_j(\mathbf{s})$ of a state $\mathbf{s} \in G_j$ is computed as $u_j(\mathbf{s}) = \mathbf{s} + \mathbf{d}_j$, where $\mathbf{d}_j \in \mathbb{Z}^n$ is a constant change vector. In many applications, a discrete state variable represents the number of instances of a certain system component type, which is incremented or decremented by a small amount. For instance, in the case of biochemical reaction networks, $\mathbf{d}_j \in \{-2, -1, \ldots, 2\}^n$, because a reaction changes the population vectors of the chemical species by an amount of at most two. Any reaction that requires the collision of more than two molecules is usually modelled as a sequence of several reactions. For the rate function α_j, we assume that the relative difference $|\alpha_j(\mathbf{s}) - \alpha_j(u(s))|/\alpha_j(\mathbf{s})$ is small for all $\mathbf{s} \in G_j$. This is the case if, for instance, α_j is linear or at most quadratic in the state variables. According to stochastic chemical kinetics, this assumption is adequate for biochemical reaction networks, because the rate of a reaction is proportional to the number of distinct combinations of reactants. Finally, we assume that the sets G_j can be represented as intersections of half planes of $\mathcal{S}$. Again, this assumption holds for biochemical reaction networks, as G_j refers to the availability of reactant molecules.

The conditions stated above ensure that we can derive geometric boundaries for the window W_r. More precisely, we can construct an n-dimensional hyper-rectangular W_r such that the left hand of Equation (4.2) is close to one. Intuitively, the boundaries of W_r describe upper and lower bounds on the state variables $s_1, \ldots, s_n$. Consider, for instance, Figure 4.2 and recall that the initial state of the process is $\mathbf{y} = (0, 0)$. For the rectangle $W =$

$\{(\mathbf{s}_R, \mathbf{s}_P) \in \mathcal{S} \mid 0 \leq \mathbf{s}_R \leq 30, 0 \leq \mathbf{s}_P \leq 120\}$, we have $P(\mathbf{X}(t) \in W, t \in [0, 1000)) \approx 0.99$.

For the construction of W_r, we use a technique that considers only the "worst case" behaviour of the Markov chain during $[t_{r-1}, t_r)$ and is therefore cheap compared to the solution of the abstract model. The random variable $\alpha_j(\mathbf{X}(t))$ represents the rate of transition type $\mathcal{C}_j$ at time t. We can assume that during a small time interval of length Δ, $\alpha_j(\mathbf{X}(t+h))$ is constant, with $0 \leq h \leq \Delta$. If $\mathbf{s}$ is the current state then the number of $\mathcal{C}_j$-transition within the next Δ time units is Poisson distributed with parameter $\alpha_j(\mathbf{s}) \cdot \Delta$ [105]. We can approximate this number by the expectation $\alpha_j(\mathbf{s}) \cdot \Delta$ of the Poisson distribution. As we are interested in an upper and lower bound, we additionally consider the standard deviation $\sqrt{\alpha_j(\mathbf{s}) \cdot \Delta}$ of the Poisson distribution. Thus, in the worst case, the number of transitions of type $\mathcal{C}_j$ is

– at least $\kappa_j^-(\mathbf{s}, \Delta) = \max(0, \alpha_j(\mathbf{s}) \cdot \Delta - \sqrt{\alpha_j(\mathbf{s}) \cdot \Delta})$,

– at most $\kappa_j^+(\mathbf{s}, \Delta) = \alpha_j(\mathbf{s}) \cdot \Delta + \sqrt{\alpha_j(\mathbf{s}) \cdot \Delta}$

Note that if, for instance, $\alpha_j(\mathbf{s}) \cdot \Delta = 1$, then we have a confidence of 91.97% that the real number of transitions lies in the interval

$$\left[\alpha_j(\mathbf{s}) \cdot \Delta - \sqrt{\alpha_j(\mathbf{s}) \cdot \Delta}, \alpha_j(\mathbf{s}) \cdot \Delta + \sqrt{\alpha_j(\mathbf{s}) \cdot \Delta}\right].$$

Let $\kappa_j \in \left\{\kappa_j^+, \kappa_j^-\right\}$ and $\mathbf{s}^{(0)} = \mathbf{s}$. For $l = 0, 1, \ldots$, the iteration

$$\mathbf{s}^{(l+1)} = \mathbf{s}^{(l)} + \sum_{j=1}^{m} \mathbf{d}_j \cdot \kappa_j(\mathbf{s}^{(l)}, \Delta) \tag{4.3}$$

yields worst-case approximations of $\mathbf{X}(t+\Delta), \mathbf{X}(t+2\Delta), \ldots$ under the condition that $\mathbf{X}(t) = \mathbf{s}$. Note that $\mathbf{s}^{(l)} \in \mathbb{R}^n_{\geq 0}$. For functions α_j that grow extremely fast in the state variables, the iteration may yield bad approximations since it is based on the assumption that the rates are constant during a small interval. In the context of biochemical reaction networks, the linearity properties mentioned above are fulfilled and Equation (4.3) yields adequate approximations. The bounds $b_i^+(\mathbf{s})$ and $b_i^-(\mathbf{s})$ for dimension $i \in \{1, \ldots, n\}$ are given by the minimal and maximal values during the iteration. More precisely, $b_i^+(\mathbf{s}) = \lceil \max_l s_i^{(l)} \rceil$ and $b_i^-(\mathbf{s}) = \lfloor \min_l s_i^{(l)} \rfloor$, where $\mathbf{s}^{(l)} = (s_1^{(l)}, \ldots, s_n^{(l)})$.

In order to construct W_r, we do not consider *all* combinations $\left\{\kappa_1^+, \kappa_1^-\right\} \times \ldots \times \{\kappa_m^+, \kappa_m^-\}$ in Equation (4.3). We choose only those combinations that do not treat preferentially transition types leading to opposite directions in the state space. Consider, for instance, Example 2.1 with $\mathbf{s} = (5, 50)$ and $\Delta = 10$.

If we assume that more reactions of type $\mathcal{C}_1$ and $\mathcal{C}_2$ happen (than on average) and fewer of $\mathcal{C}_3$ and $\mathcal{C}_4$, we get $\kappa_1^+(\mathbf{s},\Delta) = c_1 \cdot 10 + \sqrt{c_1 \cdot 10} = 1.2$, $\kappa_2^+(\mathbf{s},\Delta) = c_2 \cdot 10 \cdot 5 + \sqrt{c_2 \cdot 10 \cdot 5} = 0.83$, $\kappa_3^-(\mathbf{s},\Delta) = \max(0, c_3 \cdot 10 \cdot 5 - \sqrt{c_3 \cdot 10 \cdot 5}) = 0$, $\kappa_4^-(\mathbf{s},\Delta) = \max(0, c_4 \cdot 10 \cdot 50 - \sqrt{c_4 \cdot 10 \cdot 50}) = 0$. This means that the number of protein and mRNA molecules increases and $\mathbf{s}^{(1)} = (6.2, 50.83)$. We do not consider the combinations that contain both κ_1^+ and κ_3^+. As $\mathcal{C}_1$ equates $\mathcal{C}_3$ and vice versa, these combinations do not result in extreme values of the state variables. For each dimension, we can identify two combinations that yield minimal and maximal values by examining the vector field of the transition classes. We refer to a chosen combination as a *branch* and fix for each transition class $\mathcal{C}_j$ a choice $\kappa_j = \kappa_j^+$ or $\kappa_j = \kappa_j^-$ for all l.

For the construction of W_r, we first need to define the significant set of states at time t_{r-1}. A very precise method would require sorting of the vector $\hat{p}_{\mathbf{s}}^{(t_{r-1})}$, which we find far too expensive. Therefore, we opt for a simpler solution where we define the set $\mathcal{S}_r = \left\{\mathbf{s} \in \mathcal{S} \,|\, \hat{p}_{\mathbf{s}}^{(t_{r-1})} > \delta\right\}$ of states significant at time t_{r-1}. Here, $\delta > 0$ is a small constant that is several orders of magnitude smaller than the desired precision. For our experimental results, we used $\delta = 10^{-10}$ and decreased this value during the iteration if $\sum_{\mathbf{s} \notin \mathcal{S}_r} \hat{p}_{\mathbf{s}}^{(t_{r-1})}$ exceeded our desired precision. For each branch, we carry out the iteration in Equation (4.3) for $\lceil h_r/\Delta \rceil$ steps with 10 different initial states randomly chosen from $\mathcal{S}_r$. This yields a cheap approximation of the behaviour of the process during the interval $[0, h_r)$. For dimension i, let b_i^+ and b_i^- denote the bounds that we obtain by merging the bounds of each branch and each randomly chosen state. We set

$$W_r = \mathcal{S}_r \cup \left\{\mathbf{s} = (s_1, \ldots, s_n) \in \mathcal{S} \,|\, b_i^- \leq s_i \leq b_i^+, 1 \leq i \leq n\right\}.$$

We choose the time steps Δ in the order of the expected residence time of the current state such that the assumption of $\alpha_j(\mathbf{X}(t))$ being constant is reasonable.

The boundaries of the window become rough if h_r is large. Therefore, for the experimental results in Section 4.2.6, we choose h_r dynamically. During the iterative computation of the bounds b_i^+ and b_i^-, we compute the size of the current window W_r. We stop the iteration if $|W_r|$ exceeds twice the size of $\mathcal{S}_r$ but not before W_r has reached a minimal size of 5000 states. By doing so, we induce a sliding of the window, which is forced to move from its previous location. It is, of course, always possible to choose a smaller value for h_r if the distribution at a specific time instant $t < t_{r-1} + h_r$ is of interest.

4.2.3 Algorithm

Algorithm 4.1 *sliding_window*

Input: TCM $\langle \mathcal{S}, \mathbf{y}, \{\mathcal{C}_1, \ldots, \mathcal{C}_m\}\rangle$, $t_1, \ldots, t_{r'}$ with $0 < t_1 < \ldots < t_{r'}$;
Output: approximations $\hat{\mathbf{p}}^{(t_1)}, \ldots, \hat{\mathbf{p}}^{(t_{r'})}$ and error ϵ.

1: $W_0 \leftarrow \{\mathbf{y}\}$; $p(\mathbf{y}) \leftarrow 1$, $\epsilon \leftarrow 0$;
2: **for** $r \in \{1, \ldots, r'\}$ **do**
3: $\quad h_r \leftarrow t_r - t_{r-1}$;
4: $\quad W_r \leftarrow$ compute_window $(p, h_r, \mathcal{C}_1, \ldots, \mathcal{C}_m)$;
5: $\quad$ Construct generator Q_r of the abstract model based on $\mathcal{C}_1, \ldots, \mathcal{C}_m$ and W_r;
6: $\quad q_{\mathbf{s}} \leftarrow \begin{cases} p_{\mathbf{s}} & \text{if } \mathbf{s} \in W_{r-1} \cap W_r, \\ \sum_{\mathbf{s} \in W_{r-1} \setminus W_r} p_{\mathbf{s}} & \text{if } \mathbf{s} = s_f, \\ 0 & \text{otherwise;} \end{cases}$
7: $\quad p \leftarrow q \cdot \exp(Q_r h_r)$; // call to any numerical solution.
8: $\quad \hat{p}_{\mathbf{s}}^{(t_r)} \leftarrow p_{\mathbf{s}}$ for $\mathbf{s} \in W_r$ and $\hat{p}_{\mathbf{s}}^{(t_r)} = 0$ otherwise;
9: $\quad \epsilon \leftarrow \epsilon + p(s_f)$;

Algorithm 4.2 *compute_window*

Input: probability distribution p, step h_r, transition classes $\mathcal{C}_1 \ldots \mathcal{C}_m$,
Output: window $W_r = \left\{b^-, b^+\right\}$.

1: // define S_r for construction of W_r
2: $S_r \leftarrow \{x \mid \mathbf{p}(x) > \delta\}$;
3: $numStates \leftarrow \min(10, size(S_r))$;
4: $A \leftarrow rand(S_r, numStates)$; // choose $numStates$ random states from S_r
5: $b^+ \leftarrow -\infty$; $b^- \leftarrow +\infty$; // initial boundaries
6: **for all** $\mathbf{s}'$ in A **do**
7: $\quad$ // run continuous determ. approximation
8: $\quad$ // on $\mathbf{s}'$ and updated boundaries
9: $\quad (b^+, b^-) \leftarrow average\_approximation(\mathbf{s}', h, b^+, b^-)$;

Algorithm 4.1 describes an iterative method to approximate $\mathbf{p}^{(t_1)}, \ldots, \mathbf{p}^{(t_{r'})}$ by vectors $\hat{\mathbf{p}}^{(t_1)}$, ..., $\hat{\mathbf{p}}^{(t_{r'})}$. We start with probability 1 in the initial state $\mathbf{y}$ (line 1). In line 4, we compute the window W_r such that most of the probability mass remains within W_r during the next h_r time units, as explained in 4.2.2 and shown in Algorithm 4.2. In line 5, we construct the generator matrix of the abstract model (the finite Markov chain with state space $W_r \cup \{s_f\}$) as seen in 4.2.1). We define the initial distribution of the abstract model in line 6 and calculate its solution in line 7. The approximation $\hat{\mathbf{p}}^{(t_r)}$ of $\mathbf{p}^{(t_r)}$ is then defined in line 8. Finally, in line 9, we add the approximation error to ϵ. A detailed error analysis is given below. Note that after the r-th loop $\epsilon = 1 - \sum_{\mathbf{s} \in W_r} p_{\mathbf{s}}$, that is, in each loop, the probability of being in s_f may increase. Thus,

$$\sum_{\mathbf{s} \in \mathcal{S}} \hat{p}_{\mathbf{s}}^{(t_1)} \geq \ldots \geq \sum_{\mathbf{s} \in \mathcal{S}} \hat{p}_{\mathbf{s}}^{(t_{r'})}. \tag{4.4}$$

The general idea of this abstraction approach is apparent from Figure 4.2, but the main difficulty is to find the states that can be neglected in step r (line

Algorithm 4.3 *average_approximation*

Input: initial state $\mathbf{s}'$, length h of the time interval, old boundaries b^+, b^-;
Output: new boundaries b^+, b^-.

```
1: for all branch ρ ∈ {1,...,2n} do
2:    x^⟨ρ⟩ ← s';  // s' is the initial state of each branch
3: time ← 0;
4: Δ ← step_size;  // set the length of time steps
5: while time < h do
6:    for each branch ρ ∈ {1,...,2n} do
7:       // compare current state variables with boundaries
8:       for i = 1 to n do
9:          if x_i^⟨ρ⟩ > b_i^+ then
10:            b_i^+ ← x_i^⟨ρ⟩;  // adjust upper bound
11:         if x_i^⟨ρ⟩ < b_i^- then
12:            b_i^- ← x_i^⟨ρ⟩;  // adjust lower bound
13:      for j ← 1 to m do
14:         // choose more/fewer transitions of type R_j
15:         // depending on branch ρ
16:         κ_j ← choose_sign(ρ, j) · α_j(x^⟨ρ⟩);
17:      x^⟨ρ⟩ ← x^⟨ρ⟩ + Σ_{j=1}^{m} d^(j) · κ_j; // update state (cf. Equation (4.3))
18:   time ← time + Δ;
```

4). In Algorithm 4.2 we give the pseudocode for the window construction as explained in 4.2.2. We compute the boundaries of the window by approximating the worst case of ten randomly chosen states of the significant region ($A \subseteq \mathcal{S}_r$). Algorithm 4.3 shows how we obtain the worst case approximation of a window that covers that states visited in $[0, h]$, knowing that at time 0 we are in state $\mathbf{s}'$. This is done by splitting that time length h in smaller length Δ, and then integrating the average behaviour plus/minus its deviation (see 4.2.2).

4.2.4 Time intervals

For the experimental results in Section 4.2.6 we compare two different time stepping mechanisms for Algorithm 4.1. We either choose equidistant time steps $h_r = h$, for all r, or we determine h_r during the construction of the window W_r. The latter method yields faster running times. Depending on the dynamics of the system, long time steps may cause three problems: (1) the window is large and the size of the matrix Q_r may exceed the working memory capacity, (2) the dynamics of the system may differ considerably during a long time step and Q_r has bad mathematical properties, (3) the window may contain states that are only significant during a much shorter time interval. If, on the other hand, the time steps are too small then many iterations of the main loop have to be carried out until the algorithm terminates. The windows overlap nearly completely,

Algorithm 4.4 *average_approximation*2

Input: initial state $\mathbf{s}'$, old boundaries b^+, b^-, significant states S_r
Output: new boundaries b^+, b^-, length h_r of the time interval,

```
1: for all branch ρ ∈ {1, ..., 2n} do
2:    x^⟨ρ⟩ ← s'; // s' is the initial state of each branch
3: h_r ← 0;
4: Δ ← step_size; // set the length of time steps
5: W_r ← S_r;
6: while |W_r| < 5000 or |W_r| < 2 * |S_r| do
7:    for each branch ρ ∈ {1, ..., 2n} do
8:       ...
9:    h_r ← h_r + Δ;
10:   W_r ← W_r ∪ states(b^-, b^+); // extend window with new boundaries.
```

Algorithm 4.5 *sliding_window*2

Input: TCM $\langle \mathcal{S}, \mathbf{y}, \{\mathcal{C}_1, \ldots, \mathcal{C}_m\}\rangle$, $t_{r'}$,
Output: Approximations $\hat{\mathbf{p}}^{(t_1)}, \ldots, \hat{\mathbf{p}}^{(t_{r'})}$ and error ϵ.

```
1: ...
2: t ← 0;
3: while t < t_r' do
4:    S_r ← significant(W_r);
5:    (W_r, h_r) ← compute_window2(p, S_r, C_1, ..., C_m);
6:    ...
7:    t ← t + h_r;
```

and even though each step may require little time, the whole procedure can be computationally expensive. One possibility is to fix the size of the windows and choose the time steps accordingly. But this does not necessarily result in short running times of the algorithm either. The reason is that the time complexity of the solution methods does not depend only on the size of the matrix representing the window but also on its mathematical properties.

The problems mentioned above can be circumvented by calculating $h_1, \ldots, h_{r'}$ during the construction of the window W_r. We do this by replacing the method *average_approximation* with its new version *average_approximation*2 (see Algorithm 4.4). The new version takes as input the set S_r of significant states at time t_{r-1} and gives as an output the length h_r of the next time interval. We run the while loop in Algorithm 4.4 until (1) the window has at least a certain size and (2) the number of states in the window exceeds twice the number of the states that are significant at time t_{r-1}. The first condition ensures that the window exceeds a certain minimum size of, say, 5000 states. The second condition ensures that the new window is just big enough to move the probability mass to a region outside of S_r. More precisely, it ensures that the sets $S_1, S_2, \ldots$ are not overlapping and that subsequent sets are located next to each other (as illustrated in Figure 4.1). Note that this ensures that the resulting window does

not contain many states that are only significant during a much shorter time interval. We also replace functions *sliding_window* and *compute_window* with *sliding_window*2 and *compute_window*2 (Algorithm 4.5). By using this new version of the window approximation algorithm *sliding_window*2 only needs as input the final time horizon $t_{r'}$ (and not the intermediate points $t_1, t_2 \dots t_{r'}$). In order to track the current time, we add to the main loop a variable representing the time elapsed so far, and the for loop in line 2 is replaced by a while loop that stops when the time elapsed so far exceeds t. Finally, the method *compute_window*2 is the same as *compute_window* where we change the parameter h from an input parameter to an output parameter and we change the call *average_approximation* with *average_approximation*2.

In Section 4.2.6, we present experimental results of the sliding window method where we choose the time steps in the way described above.

Numerical Solution Methods. For the solution step in line 7 of Algorithm 4.1, we apply a numerical method to compute the matrix exponential. If Q_r is small then the matrix exponential can be computed efficiently using, for instance, Padé approximation [6, 80]. If the size of Q_r is large but Q_r is sparse then iterative methods perform better, such as uniformization [46, 64], approximations in the Krylov subspace [89], or numerical integration [48, 49].

For the experimental results that we present here, we concentrate on the uniformization method (see Chapter 4.3) and on the Krylov subspace method(see Section 5.3).

4.2.5 Error analysis

Recall that if Q is the generator matrix of the original Markov chain, $\exp(Qh_r)$ is the transition probability matrix for time step h_r. Let Q_r be the generator matrix of the abstract Markov chain constructed in the r-th step (see Algorithm 4.1, line 5). For $\mathbf{s}, \mathbf{s}' \in W_r$, we use the approximation

$$\begin{aligned}(\exp(Qh_r))_{\mathbf{s},\mathbf{s}'} &= P\left(\mathbf{X}(t_r) = \mathbf{s}' \,|\, \mathbf{X}(t_{r-1}) = \mathbf{s}\right) \\ &\approx P\left(\mathbf{X}(t_r) = \mathbf{s}' \wedge \mathbf{X}(h) \in W_r, h \in (t_{r-1}, t_r) \,|\, \mathbf{X}(t_{r-1}) = \mathbf{s}\right) \\ &= (\exp(Q_r h_r))_{\mathbf{s},\mathbf{s}'} \,. \end{aligned} \tag{4.5}$$

in line 5 of Algorithm 4.1. Thus, we ignore the probability to reach $\mathbf{s}'$ from $\mathbf{s}$ after h_r time units by leaving W_r at least once.

For the error analysis, we assume that the vector $q^{(r)}$ of size $|W_r| + 1$ is such that $q^{(r)}_{\mathbf{s}} = p^{(t_{r-1})}_{\mathbf{s}}$ if $\mathbf{s} \in W_r$. This is true for $r = 1$ and for $r > 1$ we

replace $p_{\mathbf{s}}^{(t_{r-1})}$ by $\hat{p}_{\mathbf{s}}^{(t_{r-1})}$ in Algorithm 4.1. In line 7 and 8, we define $\hat{p}_{\mathbf{s}'}^{(t_r)} = \left(q^{(r)} \cdot \exp(Q_r h_r)\right)_{s'}$ for $\mathbf{s}' \in W_r$. Thus,

$$\begin{aligned}
\hat{p}_{\mathbf{s}'}^{(t_r)} &= \left(q^{(r)} \cdot \exp(Q_r h_r)\right)_{s'} \\
&= \sum_{\mathbf{s} \in W_r} p_{\mathbf{s}}^{(t_{r-1})} \left(\exp(Q_r h_r)\right)_{\mathbf{s},\mathbf{s}'} \\
&\approx \sum_{\mathbf{s} \in W_r} p_{\mathbf{s}}^{(t_{r-1})} \left(\exp(Q h_r)\right)_{\mathbf{s},\mathbf{s}'} \\
&= \sum_{\mathbf{s} \in W_r} P\left(\mathbf{X}(t_{r-1}) = \mathbf{s}\right) \cdot P\left(\mathbf{X}(t_r) = \mathbf{s}' \,|\, \mathbf{X}(t_{r-1}) = \mathbf{s}\right) \\
&\approx \sum_{\mathbf{s} \in \mathcal{S}} P\left(\mathbf{X}(t_{r-1}) = \mathbf{s}\right) \cdot P\left(\mathbf{X}(t_r) = \mathbf{s}' \,|\, \mathbf{X}(t_{r-1}) = \mathbf{s}\right) \\
&= P\left(\mathbf{X}(t_r) = \mathbf{s}'\right) = p_{\mathbf{s}'}^{(t_r)}. \qquad (4.6)
\end{aligned}$$

The first approximation is due to Equation (4.5). The second approximation comes from the fact that we ignore the probability of not being in W_r at time t_{r-1}. In both cases we use an underapproximation. By setting $\hat{p}_{\mathbf{s}'}^{(t_r)} = 0$ if $\mathbf{s}' \notin W_r$, we obtain $\hat{p}_{\mathbf{s}'}^{(t_r)} \leq p_{\mathbf{s}'}^{(t_r)}$ for all $\mathbf{s}' \in \mathcal{S}$. Overall, we use three approximations, where probability is "lost" namely,

(a) the probability that is lost due to the approximation given by Equation (4.5),

(b) the probability of not starting in W_r at time t_{r-1} (second approximation in Equation (4.6)),

(c) the probability of leaving W_r during $[t_{r-1}, t_r)$ (which arises due to the approximation $p_{\mathbf{s}'}^{(t_r)} \approx 0$ if $\mathbf{s}' \notin W_r$).

It is easy to see that, if the probability of being in W_r during $[t_{r-1}, t_r)$ is at least $1 - \epsilon_r$ (see Equation (4.2)), then all three errors are at most ϵ_r. Thus, $||\mathbf{p}^{(t_r)} - \hat{\mathbf{p}}^{(t_r)}||_1 \leq \epsilon_r$. Note that the entry $\epsilon = p(s_f)$ that is computed in line 9 of Algorithm 4.1 contains all three approximation errors (a), (b), (c). After the termination of the for loop, ϵ contains the total approximation error, which is at most $\epsilon_1 + \ldots + \epsilon_r$.

If the approximation error ϵ in Algorithm 4.1 exceeds the desired error threshold, the window construction can be repeated using a larger window W_r. This may happen if the confidence of the estimated interval $[\kappa_j^-(\mathbf{s}, \Delta), \kappa_j^+(\mathbf{s}, \Delta)]$ for the number of transitions of type j is not large enough. In this case, the approximation $\hat{\mathbf{p}}^{(t_r)}$ can be used to determine where to expand W_r. Several heuristics for the window expansion are possible. The smooth distribution of

the probability mass, however, suggests to expand only those boundaries of W_r where states with a high probability are located.

4.2.6 Case Studies

We coded Algorithm 4.5 in C++ and ran experiments on a 3.16 GHz Intel dual-core Linux PC. We discuss experimental results that we obtained for the Example 2.1, as well as a simple enzyme example, Goutsias' model [44] and a bistable toggle switch [37]. Goutsias' model describes the transcription regulation of a repressor protein in bacteriophage λ and involves six different species and ten reactions. The bistable toggle switch is a prototype of a genetic switch with two competing repressor proteins and four reactions. All results are listed in Table 4.1.

As explained in detail below, we also implemented the method proposed by Burrage et al. [12] in order to compare it to our algorithm in terms of running time and accuracy. Moreover, for finite examples we compare our method to a global analysis, i.e. where in each step the entire state space is considered. We do not compare our method to Gillespie simulation or approximation methods based on the Fokker-Planck equation. The former method provides only estimates of the probability distribution and becomes infeasible if small probabilities are estimated [22]. The latter type of methods do not take into account the discreteness of the molecule numbers and are known to provide bad approximations in the case of small populations as considered here [34].

Parameters. We fixed the error tolerance for the numerical solutions to $\epsilon_n = 10^{-8}$ of Algorithm 4.5 for all experiments. We chose the input δ in a dynamical fashion to ensure that in the r-th step we do not lose more probability than $10^{-5} \cdot h_r/(t_{r'} - t_0)$ by restricting to significant states, that is, we decrease δ until after line 4 of Algorithm 4.1 the set S_r contains at most $10^{-5} \cdot \frac{h_r}{t_r - t_0}$ less probability than the former set S_{r-1}. In Table 4.1, we list the average value that we used for δ.

In the sequel, we give details about the parameters used for the results that we obtained for Example 4.1 and Example 2.1. For the remaining two examples, we list the corresponding chemical reactions and the parameters that we chose for the results in Table 4.1.

Example 4.1 (**Enzyme example).** *We describe an* enzyme-catalyzed substrate conversion *by the three reactions* $R_1 : E+S \to ES$, $R_2 : ES \to E+S$, $R_3 : ES \to E + P$. *This network involves four chemical species, namely, enzyme* (E), *substrate* (S), *complex* (ES), *and product* (P) *molecules. The change vec-*

tors are $v^{(1)} = (-1, -1, 1, 0)$, $v^{(2)} = (1, 1, -1, 0)$, *and* $v^{(3)} = (1, 0, -1, 1)$. *For* $(x_1, x_2, x_3, x_4) \in \mathbb{N}_0^4$, *the propensity functions are*

$$\alpha_1(x_1, x_2, x_3, x_4) = c_1 \cdot x_1 \cdot x_2, \quad \alpha_2(x_1, x_2, x_3, x_4) = c_2 \cdot x_3, \quad \alpha_3(x_1, x_2, x_3, x_4) = c_3 \cdot x_3.$$

As above, the set of states reachable from the initial state $\mathbf{y} = (y_1, y_2, y_3, y_4)$ *is finite because of the conservation laws* $y_1 = x_1 + x_3$ *and* $y_2 = x_2 + x_3 + x_4$, *where we assume that* $y_3 = y_4 = 0$.

We tried different parameter sets, referred to as pset a)-c), for Example 4.1 (see Table 4.1). For parameter combination a) we have $c_1 = c_2 = 1, c_3 = 0.1$ and start with 1000 enzymes and 100 substrates. In this case the number of reachable states is 5151. For parameter set b) and c) we have $c_1 = c_2 = c_3 = 1$ and and start with 100 enzymes and 1000 substrates and 500 enzymes and 500 substrates, which yields 96051 and 125751 reachable states, respectively. Each time we choose the time horizon according to the time until most of the probability mass is concentrated in the state in which all substrate molecules are transformed into products. For the time steps h_r in Algorithm 4.1, we apply the condition described in Section 4.2.4.

We consider four branches for the iteration in Equation (4.3) in order to determine upper and lower bounds on the state variables. (1) To obtain an estimate for the maximal number of complex molecules (and a minimum for the enzyme population), we enforce more reactions of type R_1 than on average ($\kappa_1 = \kappa_1^+$), and fewer of types R_2 and R_3 ($\kappa_3 = \kappa_3^-$ and $\kappa_2 = \kappa_2^-$). (2) By considering fewer reactions of type R_1 ($\kappa_1 = \kappa_1^-$), and more of types R_2 and R_3 ($\kappa_3 = \kappa_3^+$ and $\kappa_2 = \kappa_2^+$) the complex population becomes minimal (and the enzyme population maximal). (3) An estimate for the minimal number of type P molecules (and the maximal number of type S molecules) is obtained by enforcing more reactions of type R_2 ($\kappa_2 = \kappa_2^+$), and fewer of types R_1 and R_3 ($\kappa_1 = \kappa_1^-$ and $\kappa_3 = \kappa_3^-$). (4) Finally, more reactions of types R_1 and R_3 ($\kappa_1 = \kappa_1^+$ and $\kappa_3 = \kappa_3^+$), and fewer of type R_2 ($\kappa_2 = \kappa_2^-$) gives a maximal increase of the number of product molecules (and minimizes the number of substrate molecules).

For the enzyme example, if the initial conditions are fixed a state is uniquely determined by at least two entries, say, the population of complex and product molecules. However, a rectangular window shape yields poor results if the expected number of complex molecules is high. The reason is that in this case the probability mass is located on a diagonal (cf. Figure 4.1). If the set of signifi-

	parameters			results					
	name of example	time horizon	δ	sliding_window + uniform.		sliding_window + Krylov	window construction		
				error	times in sec		perc.	average wind. size	
1	Enzyme (pset a)	70	10^{-8}	1.4×10^{-5}	6	5	1%	977	
2	Enzyme (pset b)	12	10^{-10}	3.3×10^{-5}	134	98	14%	4777	
3	Enzyme (pset c)	5	10^{-10}	3.5×10^{-7}	8	6	37%	5038	
4	Gene (pset a)	10^4	10^{-10}	1.6×10^{-5}	103	102	36%	32248	
5	Gene (pset b)	10^4	10^{-10}	1.8×10^{-5}	137	123	32%	38282	
6	Goutsias' model	300	10^{-11}	7.6×10^{-5}	15943	8412	15%	538815	
7	Toggle switch	10^4	10^{-15}	2.7×10^{-5}	31	10	1%	63001	

Table 4.1: Parameters and results of the sliding window method

cant states is captured by a rectangular window it may contain many states that are not significant. This problem can be circumvented by considering bounds for all state variables during the window construction as well as the conservation laws. More precisely, the parallelogram in Figure 4.1 are constructed by computing for each value $x_4 \in [b_4^-, b_4^+]$ of P upper and lower bounds on ES by $\min\left\{b_3^+, y_1 - b_1^-, y_2 - x_4 - b_2^-\right\}$ and $\max\left\{b_3^-, y_1 - b_1^+, y_2 - x_4 - b_2^+\right\}$, where $\mathbf{y} = (y_1, y_2, 0, 0)$ is the initial population vector and $b^+ = (b_1^+, b_2^+, b_3^+, b_4^+)$ and $b^- = (b_1^-, b_2^-, b_3^-, b_4^-)$ are the upper and lower bounds on the populations of E, S, ES, and P.

Note that the parallelogram in Figure 4.1 was induced by the conservation laws of the system. In general, conservation laws should be taken into account since otherwise the window may be inconsistent with the conservation laws, i.e. it may contain states that are not reachable.

Gene expression example. In Table 4.1 we present results for Example 2.1. The difference between parameter set a) and parameter set b), referred to as pset a) and pset b), is that for a) we start with the empty system and for b) we start with 100 mRNA molecules and 1000 proteins. For both variants, we choose rate constants $c_1 = 0.5$, $c_2 = 0.0058$, $c_3 = 0.0029$, $c_4 = 0.0001$. The time steps that we use are determined by the condition in Section 4.2.4. Note that we cannot solve this example using a global method because the number of reachable states is infinite. The column *error* contains the total error ϵ and *times in sec* refers to the running time in seconds. In column *perc.* we list the percentage of the total running time that was spent for the window construction. The column *average wind. size* refers to the average number of states in the window.

For the gene expression example, we use four branches: We maximize the number of mRNA molecules by choosing κ_1^+ and κ_3^- and minimize it with κ_1^- and κ_3^+. Reactions R_2 and R_4 are irrelevant for this species. We maximize the protein population by choosing κ_1^+, κ_2^+, κ_3^-, and κ_4^- and minimize it with κ_1^-, κ_2^-, κ_3^+, and κ_4^+.

Example 4.2 (Goutsias' model). *In [44], Goutsias defines a model for the transcription regulation of a repressor protein in bacteriophage λ. This protein is responsible for maintaining lysogeny of the λ virus in E. coli. The model involves 6 different species and the following 10 reactions. We list the reactions and rate constants in Table 4.2 (see [12, 44]).*

We used the same kinetic constants as Goutsias [44] and Sidje et al. [12], as well as the same initial state. Below, we list the branches for upper bounds

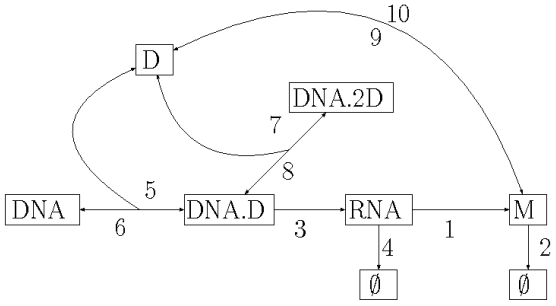


Figure 4.3: Dependencies between the reactions of Goutsias' model.

$RNA \xrightarrow{c_1} RNA + M$	$c_1 = 0.043$	production of protein
$M \xrightarrow{c_2} \emptyset$	$c_2 = 7\text{e}{-4}$	degradation of protein
$DNA.D \xrightarrow{c_3} RNA + DNA.D$	$c_3 = 0.072$	production of mRNA
$RNA \xrightarrow{c_4} \emptyset$	$c_4 = 4\text{e}{-3}$	degradation of mRNA
$DNA + D \xrightarrow{c_5} DNA.D$	$c_5 = 0.02$	1st dimer binding
$DNA.D \xrightarrow{c_6} DNA + D$	$c_6 = 0.48$	1st dimer unbinding
$DNA.D + D \xrightarrow{c_7} DNA.2D$	$c_7 = 2\text{e}{-4}$	2nd dimer binding
$DNA.2D \xrightarrow{c_8} DNA.D + D$	$c_8 = 9\text{e}{-12}$	2nd dimer unbinding
$M + M \xrightarrow{c_9} D$	$c_9 = 0.083$	dimerization
$D \xrightarrow{c_{10}} M + M$	$c_{10} = 0.5$	dissociation of dimer

Table 4.2: List of reactions of the phage λ model.

on the state variables. Lower bounds are obtained if the opposite combination is considered, respectively. We refer to Figure 4.3 for an illustration of the dependencies between the reactions that simplifies the choice of the branches. We maximize the RNA population by choosing the combination κ_1^-, κ_2^-, κ_3^+, κ_4^-, κ_5^+, κ_6^-, κ_7^-, κ_8^+, κ_9^+, κ_{10}^-. We maximize the monomer population by choosing the combination κ_1^+, κ_2^-, κ_3^+, κ_4^-, κ_5^+, κ_6^-, κ_7^-, κ_8^+, κ_9^-, κ_{10}^+. We maximize the number of dimer molecules by choosing the combination κ_1^+, κ_2^-, κ_3^+, κ_4^-, κ_5^+, κ_6^-, κ_7^-, κ_8^+, κ_9^+, κ_{10}^-. Note that although dimers are consumed by reaction 5, choosing κ_5^+ maximizes the number of dimers in the system. This is because reaction 5 is necessary to produce monomers and therefore also dimers.

We never run out of memory with the sliding window method, but the running times can be huge for a long time horizon. The reason is that the windows are large since the system contains many monomers and dimers at later time instances. For the results in Table 4.1 we considered the system till

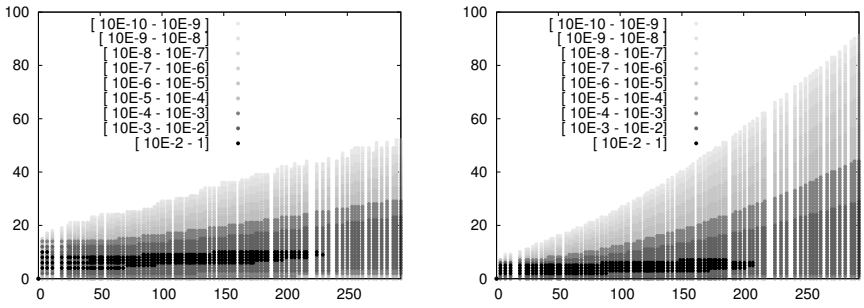


Figure 4.4: The probability distribution of monomers (left) and dimers (right) during the time interval $[0, 300)$.

time $t = 300$, whereas for Sidje et al. [12], the longest time horizon is $t = 100$. In Figure 4.4 we plot the distribution of the species M and D.

Bistable toggle switch. The toggle switch involves two chemical species A and B and four reactions. Let $x = (x_1, x_2) \in \mathbb{N}_0^2$. The reactions are $\emptyset \to A$, $A \to \emptyset$, $\emptyset \to B$, $B \to \emptyset$ and their propensity functions $\alpha_1, \ldots, \alpha_4$ are given by $\alpha_1(x) = c_1/(c_2 + x_2^\beta)$, $\alpha_2(x) = c_3 \cdot x_1$, $\alpha_3(x) = c_4/(c_5 + x_1^\gamma)$, $\alpha_4(x) = c_6 \cdot x_2$. Note that in this example the propensity functions are not of the form described in Equation (2.1). For our experimental results, we chose the same parameters as Sjöberg et al. [104], that is, $c_1 = c_4 = 3 \cdot 10^3$, $c_2 = c_5 = 1.1 \cdot 10^4$, $c_3 = c_6 = 0.001$, and $\beta = \gamma = 2$. The initial distribution is a Gaussian distribution $\mathcal{N}(\mu, \sigma^2)$ with $\mu = (133, 133)^T$ and $\sigma = (\sqrt{133}, \sqrt{133})$. We consider the obvious four branches each of which is intended to minimize/maximize one of the two components. The branch minimizing A for example will have less of the first reaction and more of the second.

Accuracy. The column labelled by *error* in Table 4.1 shows the total error ϵ of the sliding window method plus the uniformization error (which is bounded by $\epsilon_n = 10^{-8}$). The error using the Krylov subspace method instead yields the same accuracy because for both, uniformization and the Krylov subspace method, the error bound is specified a priori. For all examples, the total error does not exceed 1×10^{-4}, which means that not more than 0.01 percent of the probability mass is lost during the whole procedure. It would, of course, be possible to add an accuracy check in the while loop of Algorithm 4.1, expand the current window if necessary, and recalculate. But as the method consistently returns a small error, this has been omitted.

We also considered relative errors, that is,

$$\frac{p_{\mathbf{s}}^{(t_{r'})} - \hat{p}_{\mathbf{s}}^{(t_{r'})}}{p^{(t_{r'})}\mathbf{s}},$$

for states $\mathbf{s} \in W_{r'}$ with $p^{(t_{r'})}(\mathbf{s}) > 10^{-5}$. We approximate the value $p_{\mathbf{s}}^{(t_{r'})}$ by solving Equation (4.8) via global uniformization, where we use truncation error $\epsilon_n = 10^{-8}$. Since this is only possible if the state space is finite, we compared relative errors only for the enzyme example. Our calculations show that the relative errors are always smaller than 10^{-4}.

In order to support our considerations in Section 4.2.2, we carried out experiments in which we exclusively chose the average in line 17 of Algorithm 4.3. More precisely, for the construction of the window we do not consider the deviations in the numbers of reactions but only the average number. In this case, we called the method *average_approximation* with input $2 \cdot h$ to make sure that on average the probability mass moves to the center of the window and not too close to the borders. For this configuration, the total error is several orders of magnitude higher, e.g., for parameter set a) of the enzyme example the total error is 0.0224.

Finally, we test the size of the windows constructed in lines 7–10 of Algorithm 4.3. We change Algorithm 4.3 by decreasing the size of the window by 5% between lines 10 and 11. In this case, the total error ϵ increases. For instance, $\epsilon = 0.35$ for parameter set a) of the enzyme example. These results substantiate that the size and the position of the sliding window is such that the approximation error is small whereas significantly smaller windows result in significantly higher approximation errors.

Running time. For the time complexity analysis, we concentrate on three main issues.

– Sliding window method vs. global analysis: We compare the sliding window method with a global solution in one step, and with another window method, where the size of the window is doubled if necessary.

– Solution method (uniformization vs. Krylov subspace method): We vary the solution method by exchanging uniformization with the Krylov subspace method (methods that will be presented later on in this thesis).

– Time intervals (equidistant vs. condition from Section 4.2.4): We use different methods to determine the length h_r of the next time step in line 3 of Algorithm 4.1.

Sliding window method vs. global analysis. We used the enzyme example to compare the sliding window solution with a global solution (global uniformization and global Krylov subspace method), since it has a finite state space. Note that all other examples cannot solved using a global method since their state space is infinite. We list the time needed for the computation of $\boldsymbol{p}^{(t_{r'})}$ (cf. Equation (2.6)) with the global method in Table 4.3. Observe that the total error of the global uniformization method is smaller (compare the columns labeled by *error*) since the only error source is the truncation of the infinite sum that we will see in Equation (4.8). In the column with heading *#states* we list the number of states that are reachable. During the global solution we consider all reachable states at all time whereas in the sliding window method the average number of states considered during a time step is much smaller. This is the main reason why the sliding window method is much faster. Moreover, in the case of uniformization, the rate for global uniformization is the maximum of all exit rates, whereas for local uniformization, we take the maximum over all states in the current window. Note that the global maximum can be large compared to the local maxima. This explains the bad performance of the global uniformization method. When the Krylov subspace method is used for a global solution, the running times of the global solutions are also higher than the times of the local Krylov subspace method (sliding window method combined with the Krylov subspace method). Again, the reason is that a smaller number of states is considered during the sliding window iteration. Moreover, the matrices Q_r have numerical properties that facilitate the use of bigger, and thus, fewer time steps. The total number of iteration steps used to solve Equation (2.8) with the Krylov subspace method and the sliding window method is indeed small when compared to the global Krylov subspace method (on average around 20 times fewer steps).

We now focus on a comparison between our sliding window method and another local method, called *doubling window method*. For the latter, we compute the probability vectors in a similar way as Sidje et al. [12]. We start with an initial window and apply the Krylov algorithm. We do not iterate over the time intervals $[t_{r-1}, t_r)$ but use the step sizes of the Krylov subspace method (cf. Section 5.3). After each time step, we remove those parts of the window that will not be used for the remaining calculations. We expand the size of the window if the error exceeds a certain threshold. Since the performance of the method depends heavily on the initial window and the directions in which a window is expanded, we start initially with the same window as the sliding window method and expand always in the directions that are most advanta-

	global solution				sliding_window			
	error	uniform.	Krylov	#states	error	uniform.	Krylov	average wind. size
Enzyme (pset a)	5.0×10^{-9}	44.1 min	4.2 min	5151	1.4×10^{-5}	6 sec	5 sec	977
Enzyme (pset b)	1.5×10^{-7}	6.4 h	2.7 h	96051	3.3×10^{-5}	2.2 min	98 sec	4777
Enzyme (pset c)	–	> 12 h	5.6 h	125751	3.5×10^{-7}	8 sec	6 sec	5038

Table 4.3: Sliding window method vs. global analysis for the finite enzyme example.

geous for the computation. For this we used information about the direction in which the probability mass is moving that we obtained from experiments with the sliding window method. The expansion of a window is realized by doubling the length of all of its edges.

We applied the doubling window method to the enzyme example and the gene expression. For all parameter sets that we tried, the sliding window method outperforms the doubling window method w.r.t. running time (with an average speed-up factor of 5). The total number of iterations of the Krylov subspace approximation is up to 13 times smaller in the case of the sliding window method compared to the doubling window method (with an average of 6.5). Note that for arbitrary systems the doubling window method cannot be applied without additional knowledge about the system, i.e., it is in general not clear, in which direction the window has to be expanded.

Our results indicate that the sliding window method achieves a significant speed-up compared to global analysis, but also compared to the doubling window method. Moreover, while global analysis is limited to finite-state systems and the doubling window methods requires additional knowledge about the system, our method can be applied to any system where the significant part of the probability mass is located at a tractable subset of states. If the dimension of the system is high, then the significant part of the probability mass may be located at intractably many states and in this case the memory requirements of our algorithm may exceed the available capacity.

Solution method. During the sliding window iteration different solution methods can be applied in line 13 of Algorithm 4.1. We concentrate on the uniformization method and on the Krylov subspace method. The running times in Table 4.1 (compare the columns labeled by *sliding_window + uniformization* with the columns labeled by *sliding_window + Krylov*) show that the Krylov subspace method performs better (average speed-up factor of around 1.5). The reason is that the Krylov subspace method is more robust to stiffness than uniformization. For non-stiff systems, uniformization is known to outperform the Krylov subspace method [94, 105]. However, since biochemical network models are typically stiff, the Krylov subspace method seems to be particularly well suited in this area.

Time intervals. In order to confirm our considerations in Section 4.2.4, we also applied the sliding window method using equidistant time steps. For all examples, using equidistant time steps results in longer computation times compared to using the condition that we presented in Section 4.2.4 (with an

average speed-up factor of 3.5). A dynamic choice of the time steps has also the advantage that we can control the size of the windows and avoid that the memory requirements of the algorithm exceed the available capacity.

4.2.7 Conclusions

In this section we have shown a method of reducing the state space of a transition class model in order to obtain an approximative, yet accurate solution of the chemical master equation. In the next two sections, we present methods that reduce the continuous-time propagation problem to a discrete-time propagation problem. The space reduction is then applied for the discrete-time problem that we obtain as shown in Chapter 3. Unlike the sliding window method, the next algorithms that we present are general to any propagation models.

However, the heuristic that we use for predicting the window is tedious, requires a lot of care and due to its box shape leads to an over-approximation of the significant region of the state space. Next we will present how the threshold abstraction can be used on the uniformization method in order to construct a "window" that accurately capture the significant region.

4.3 Uniformization

The uniformization method was introduce in the setting of Markov chains by Jensen [64] and it is also referred to as Jensen's method, randomization, or discrete-time conversion. In the performance analysis of computer systems, this method is popular and often preferred over other methods, such as Krylov subspace methods and numerical integration methods [94, 105]. Recently, uniformization has also been used for the solution of the CME [53, 100, 111].

The method is used to relate the solution of a continuous-time Markov chain to that of one or more discrete-time Markov chains, thus reducing a continuous-time problem to discrete-time. Here, we generalize from Markov chains to propagation models, thus reducing continuous-time propagation problems to the discrete-time case, which was already addressed in Chapter 3. This generalization is exact for linear propagation models, such as those that represent a chemical master equation(see 2.3.3), and approximative for non-linear ones.

4.3.1 Standard Uniformization (SU)

Let $N = \langle \mathcal{S}, \mathcal{M}, \zeta, \pi \rangle$ be a PM for which we want to evaluate the continuous-time propagation process $\boldsymbol{g}$ at time t. The basic idea of uniformization is to define a second propagation model N^u and a Poisson process $\big(X^P(t)\big)_{t \geq 0}$ with rate $\Lambda \cdot t$ and with probability propagation process $\mathcal{P}_{(\Lambda \cdot t)}$ such that:

$$\boldsymbol{g}^{(t)} = \sum_{k=0}^{\infty} \mathcal{P}_{(\Lambda \cdot t)}(k) \cdot \boldsymbol{w}^{(k)},$$

where $\boldsymbol{w} = \boldsymbol{f}^u$ is the propagation process of N^u. In other words, the discrete-time propagation process of N^u (u comes from "uniformization") is related with the continuous-time propagation process of N.

First, we present the standard uniformization method for linear propagation models, and then we show the approximation that can be applied for non-linear models.

SU for linear propagation models. Recall that for linear propagation models there exists a matrix Q that encodes the edge function π (see Lemma 2.2). Let $\lambda_s = \sum_{\mathbf{s}' \neq \mathbf{s}} Q(\mathbf{s}, \mathbf{s}')$ be the *exit rate* of state $\mathbf{s} \in \mathcal{S}$, and let I be the identity matrix. An *uniformization rate* is a value Λ such that $\Lambda \geq \max_{\mathbf{s} \in \mathcal{S}} \lambda_s$. Given an uniformization rate Λ we construct the transition matrix

$$P = I + \frac{1}{\Lambda} \cdot Q,$$

and we let N^u be the propagation model $\langle \mathcal{S}, \mathcal{M}, \zeta, \pi^u \rangle$, where π^u is the edge function that corresponds to the matrix P. Note that a diagonal entry in P defines the self-loop propagation $1 - \lambda_s/\Lambda$ of a state $\mathbf{s}$, which is nonzero if and only if $\Lambda > \lambda_s$. For $k \geq 1$, the matrix P^k contains the k-step transition propagations and, as $\boldsymbol{w}^{(0)} = \zeta$ is the initial mass distribution of N^u, the vector $\boldsymbol{w}^{(k)} = \boldsymbol{w}^{(0)} \cdot P^k$ contains the discrete-time mass after k steps in N^u.

We now show that the solution of the continuous-time process $\boldsymbol{g}$ at time t is equivalent with that of the discrete-time process $\boldsymbol{f}^u = \boldsymbol{w}$, where the number of discrete steps of $\boldsymbol{f}^u$ has a Poisson distribution with parameter $\Lambda \cdot t$, i.e.,

$$Pr(k \text{ steps until time } t) = Pr\big(X^P(t) = k\big) = \mathcal{P}_{(\Lambda \cdot t)}(k) = e^{-\Lambda t} \cdot \frac{(\Lambda t)^k}{k!}. \quad (4.7)$$

The solution for $\boldsymbol{g}$, in Equation (2.10), can be developed as [21, 45, 105]

$$\begin{aligned}\boldsymbol{g}^{(t)} &= \boldsymbol{g}^{(0)} \cdot \sum_{k=0}^{\infty} \frac{(Qt)^k}{k!} \\ &= \boldsymbol{g}^{(0)} \cdot \sum_{k=0}^{\infty} e^{-\Lambda t} \cdot \frac{(\Lambda t)^k}{k!} \cdot P^k \\ &= \sum_{k=0}^{\infty} e^{-\Lambda t} \cdot \frac{(\Lambda t)^k}{k!} \cdot \boldsymbol{w}^{(k)} \\ &= \sum_{k=0}^{\infty} \mathcal{P}_{(\Lambda \cdot t)}(k) \cdot \boldsymbol{w}^{(k)} \end{aligned} \tag{4.8}$$

Equation (4.8) has nice properties compared to Equation (2.10). There are no negative summands involved from the way Λ was chosen. Moreover, $\boldsymbol{w}^{(k)}$ can be computed inductively by

$$\boldsymbol{w}^{(0)} = \boldsymbol{g}^{(0)}, \quad \boldsymbol{w}^{(j)} = \boldsymbol{w}^{(j-1)} \cdot P, \quad j \in \{1, 2, \ldots\}\,. \tag{4.9}$$

As shown in the previous chapter, if P is sparse, $\boldsymbol{w}^{(k)}$ can be calculated efficiently even if the size of the state space is large.

Lower and upper summation bounds L and U for the number of steps k can be obtained such that for each state $\mathbf{s}$ the truncation error [35]

$$\begin{aligned}\boldsymbol{g}_{\mathbf{s}}^{(t)} - \sum_{k=L}^{U} e^{-\Lambda t} \cdot \frac{(\Lambda t)^k}{k!} w_{\mathbf{s}}^{(k)} &= \sum_{\substack{0 \le k < L, \\ U < k < \infty}} e^{-\Lambda t} \cdot \frac{(\Lambda t)^k}{k!} w_{\mathbf{s}}^{(k)} \\ &\le \sum_{\substack{0 \le k < L, \\ U < k < \infty}} e^{-\Lambda t} \cdot \frac{(\Lambda t)^k}{k!} \\ &= 1 - \sum_{k=L}^{U} e^{-\Lambda t} \cdot \frac{(\Lambda t)^k}{k!} < \epsilon \end{aligned} \tag{4.10}$$

can be a priori bounded by a predefined error tolerance $\epsilon > 0$. Thus, $\boldsymbol{g}^{(t)}$ can be approximated with arbitrary accuracy by

$$\boldsymbol{g}^{(t)} \approx \sum_{k=L}^{U} e^{-\Lambda t} \cdot \frac{(\Lambda t)^k}{k!} \cdot \boldsymbol{w}^{(k)} \tag{4.11}$$

as long as the required number of summands is not extremely large.

SU for non-linear propagation models. So far we have seen how to solve the CTPP problem for linear propagation models. Here, we will give an

uniformization method for propagation models for which the following linear approximation is acceptable:

$$\sum_{k=0}^{\infty} c_k \cdot \pi_{\mathbf{s}\to\mathbf{s}'}(\Lambda_k) \approx \pi_{\mathbf{s}\to\mathbf{s}'}(\sum_{k=0}^{\infty} c_k \cdot \Lambda_k). \tag{4.12}$$

Note that this approximation is exact for propagation models that represent the chemical master equation (where $\pi_{\mathbf{s}\to\mathbf{s}+\mathbf{d}_j}(p) = p\cdot\alpha_j(\mathbf{s})$), but that accepting this approximation does not imply that a matrix notation is possible as in the case of linear propagation models.

Theorem 4.1. *For a PM $N = \langle \mathcal{S}, \mathcal{M}, \zeta, \pi\rangle$ and a time horizon t, let Λ be a uniformization rate such that*

$$\Lambda \geq \pi_{\mathbf{s}\to\mathbf{s}'}(\mu), \text{ for all } \mathbf{s},\mathbf{s}' \in \mathcal{S}, \mu = g_{\mathbf{s}}^{(h)}, h \leq t.$$

Let N^a be a second PM with $N^a = (\mathcal{S}, \mathcal{M}, \zeta, \pi^a)$, where $\pi^u = \frac{1}{\Lambda}\pi$, then, if:

$$\sum_{k=0}^{\infty} c_k \cdot \pi_{\mathbf{s}\to\mathbf{s}'}(\Lambda_k) = \pi_{\mathbf{s}\to\mathbf{s}'}(\sum_{k=0}^{\infty} c_k \cdot \Lambda_k), \tag{4.13}$$

if $\mathbf{w} = \boldsymbol{f}^a$ we have that:

$$g_{\mathbf{s}'}^{(t)} = \sum_{k=0}^{\infty} \mathcal{P}_{(\Lambda\cdot t)}(k) \cdot w_{\mathbf{s}'}^{(k)}. \tag{4.14}$$

Proof. We will show that $\sum_{k=0}^{\infty} \mathcal{P}_{(\Lambda\cdot t)}(k) \cdot w_{\mathbf{s}'}^{(k)}$ verifies the differential equation(2.9). To do so, we compute its derivative:

$$\begin{aligned}
\frac{d\sum_{k=0}^{\infty} \mathcal{P}_{(\Lambda\cdot t)}(k) \cdot w_{\mathbf{s}'}^{(k)}}{dt} &= \sum_{k=0}^{\infty} \frac{d\mathcal{P}_{(\Lambda\cdot t)}(k)}{dt} \cdot w_{\mathbf{s}'}^{(k)} \\
&= \sum_{k=0}^{\infty} \Lambda \cdot (\mathcal{P}_{(\Lambda\cdot t)}(k-1) - \mathcal{P}_{(\Lambda\cdot t)}(k)) \cdot w_{\mathbf{s}'}^{(k)} \\
&= \sum_{k=0}^{\infty} \Lambda \cdot \mathcal{P}_{(\Lambda\cdot t)}(k) \cdot (w_{\mathbf{s}'}^{(k+1)} - w_{\mathbf{s}'}^{(k)}) \\
&= \sum_{k=0}^{\infty} \Lambda \cdot \mathcal{P}_{(\Lambda\cdot t)}(k) \cdot \left(\sum_{\mathbf{s}\in\mathcal{S}} \pi^a_{\mathbf{s}\to\mathbf{s}'}(w_{\mathbf{s}}^{(k)}) - \sum_{\mathbf{s}\in\mathcal{S}} \pi^a_{\mathbf{s}'\to\mathbf{s}}(w_{\mathbf{s}'}^{(k)}) + \pi^a_{\mathbf{s}'\to\mathbf{s}'}(w_{\mathbf{s}'}^{(k)}) \right) \\
&= \sum_{\mathbf{s}\in\mathcal{S}} \pi_{\mathbf{s}\to\mathbf{s}'}(g_{\mathbf{s}}^{(t)}) - \sum_{\mathbf{s}\in\mathcal{S}} \pi_{\mathbf{s}'\to\mathbf{s}}(g_{\mathbf{s}'}^{(t)}) + \pi_{\mathbf{s}'\to\mathbf{s}'}(g_{\mathbf{s}'}^{(t)}) \\
&= \frac{dg_{\mathbf{s}'}^{(t)}}{dt}.
\end{aligned}$$

□

As before, truncation points L and R, for the iteration step k can be obtained here during the computation of the solution $\mathbf{g}^{(t)}$.

Time complexity and stiffness. As $\Lambda \cdot t$ grows, the Poisson distribution flattens and the left truncation point L in Equation (4.11) grows linearly in $\Lambda \cdot t$, while the number of significant Poisson probability terms is $O(\sqrt{\Lambda \cdot t})$ [35]. If the vectors $\boldsymbol{w}^{(L)}, \boldsymbol{w}^{(L+1)}, \ldots, \boldsymbol{w}^{(U)}$ are computed using U matrix-vector multiplications (cf. Equation (4.9)), then the complexity of the uniformization procedure is $O(\nu \cdot \Lambda \cdot t)$ where ν is the number of nonzero elements in P.

All analysis methods (simulation-based or not) encounter serious difficulties if the underlying model is *stiff*. In a stiff model the components of the underlying system act on time scales that differ by several orders of magnitude and this arises in various application domains, especially in systems biology. For a stiff model, the uniformization rate $\Lambda \geq \max_{\mathbf{s} \in \mathcal{S}} \lambda_s$ will correspond to the fastest time scale. By contrast, a significant change of the slow components can be observed only during a period of time that corresponds to the slowest time scale. The uniformization method is then extremely time consuming because of a very large *stiffness index* $t \cdot \max_{\mathbf{s} \in \mathcal{S}} \lambda_s$ [26].

By using methods from Chapter 3 or from Section 4.2, uniformization can be applied in a local fashion such that stiffness has a less negative effect on the performance of the method. So, either sliding window or threshold abstraction enable uniformization to perform well even for stiff systems and when standard uniformization is used in combination with the threshold abstraction and the on-the-fly state space construction, we call the resulting algorithm *fast standard uniformization* (FSU). In addition, we can overcome the drawback of stiffness by using adaptive uniformization, which is presented next.

4.3.2 Adaptive Uniformization (AU)

Adaptive uniformization overcomes the drawback related to stiffness mentioned above by replacing the Poisson process $(X^P(t))_{t \geq 0}$ with a *birth process* [33] $(X^B(t))_{t \geq 0}$. Intuitively, the clock X^B runs at a slower speed than X^P and has fewer jumps within the time interval $[0, t)$. Therefore AU requires fewer terms in the truncated sum in Equation (4.11), the downside being that the birth process is more expensive to solve than the Poisson process.

AU for linear propagation models. Consider a linear propagation model $N = \langle \mathcal{S}, \mathcal{M}, \zeta, \pi \rangle$, and let Q be the generator matrix that encodes the edge function π. For $\mathcal{S}_k$ a subset of $\mathcal{S}$, we define:

$$\begin{aligned}
&\text{generator matrix} && Q_k(\mathbf{s}, \mathbf{s}') = Q(\mathbf{s}, \mathbf{s}') && \mathbf{s} \in \mathcal{S}_k, \mathbf{s}' \in \mathcal{S} \\
& && Q_k(\mathbf{s}, \mathbf{s}') = 0 && \text{otherwise} \\
&\text{adaptive uniformization rate} && \Lambda_k = \max_{\mathbf{s} \in \mathcal{S}_k} \lambda_s \\
&\text{stochastic transition mastrix} && P_k = I + \frac{1}{\Lambda_k} \cdot Q_k.
\end{aligned}$$

For $k = 0, 1, \ldots$ we inductively define a sequence $\mathcal{S}_0, \mathcal{S}_1, \ldots$ of subsets of $\mathcal{S}$ from a sequence of row vectors $\boldsymbol{w}^{(0)}, \boldsymbol{w}^{(1)}, \ldots$. Let $\mathcal{S}_0 = \{\mathbf{s} \,|\, \zeta_{\mathbf{s}} > 0\}$ and let $w^{(0)} = \boldsymbol{g}^{(0)}$. For $k = 0, 1, \ldots$, we define:

$$\boldsymbol{w}^{(k+1)} = \boldsymbol{w}^{(k)} \cdot P_k \tag{4.15}$$

and

$$\mathcal{S}_{k+1} = \left\{ \mathbf{s} \in \mathcal{S} \,|\, w_{\mathbf{s}}^{(k+1)} > 0 \right\}. \tag{4.16}$$

We define the birth process $\left(X^B(t)\right)_{t \geq 0}$ by the time-independent transition probabilities

$$Pr\big(X^B(t + dt) = k + 1 \,|\, X^B(t) = k\big) = \Lambda_k \cdot dt,$$

where $[t, t + dt)$ is an infinitesimal time interval.

Example 4.3 (AU and SU comparison)**.** *The construction of* $\mathbf{X}^u(l)$ *and of the Poisson process* $X^P(t)$ *is illustrated in Figure 4.5 by means of a very simple CTMC. We use a graphical description of CTMCs and DTMCs where the nodes of the graph correspond to the states of the process and the edges are labelled by the entries of the associated generator matrix in the case of CTMCs and by transition probabilities in the case of*

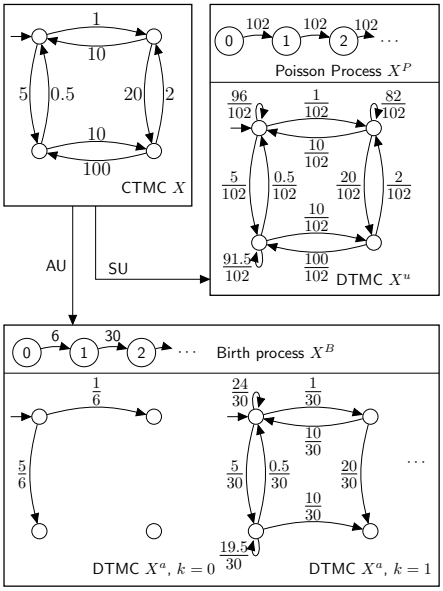


Figure 4.5: Standard and adaptive uniformization of a simple CTMC.

DTMCs. For SU, the original process $\mathbf{X}(t)$ *(upper left box) is split into the DTMC* $\mathbf{X}^u(k)$ *(lower part of the upper right box) and the Poisson process with rate* $\Lambda = 102$ *(upper part of the upper right box). The box at the bottom shows the first two steps of adaptive uniformization of the same CTMC* $\mathbf{X}(t)$*.*

Let N^a be the propagation model with step-dependent one-step transition probability matrices $P_0, P_1, \ldots$ and initial distribution ζ. Formally we have that $N^a = \langle \mathcal{S}, \mathbb{N}_0 \times [0,1], \zeta^a, \pi^a \rangle$, where

$$\zeta^a{}_{\mathbf{s}} = \langle 0, \zeta_{\mathbf{s}} \rangle , \quad \mathbf{s} \in \mathcal{S}$$
$$\pi^a_{\mathbf{s}\to\mathbf{s}'}(\langle k, \Lambda \rangle) = \langle 1, \Lambda \cdot P_k(\mathbf{s}, \mathbf{s}') \rangle , \quad k \in \mathbb{N}_0, \Lambda \in [0,1], \mathbf{s}, \mathbf{s}' \in \mathcal{S}.$$

We note that the mass space allows for the propagation of probabilities and of the step count k, which is incremented in each step by 1. The vector $\boldsymbol{w}^{(0)}$ is the initial distribution of N^a and $\boldsymbol{f}^{a(k)}$ contains the mass distribution of N^a after k steps, which is the same as $\boldsymbol{w}^{(k)}$ as defined in Equation (4.15). Van Moorsel showed that

$$\boldsymbol{g}^{(t)} = \sum_{k=0}^{\infty} \boldsymbol{w}^{(k)} \cdot Pr\big(X^B(t) = k\big) , \tag{4.17}$$

provided that N is describing a chemical master equation and that X^B does not explode[1] [109]. As we will show below, as a consequence of our proof for non-linear models, for non-linear propagation models, this equation also holds when N is a general linear propagation model.

We let $\boldsymbol{\Lambda}_{(k)}$ denote a series with elements Λ_k, and we define the function

$$\mathcal{B}_{\boldsymbol{\Lambda}_{(k)} \cdot t}(k) = Pr\big(X^B(t) = k\big) .$$

[1]The process X^B is said to explode iff the sum of the average residence times in the visited states converges, i.e., $\sum_{k\geq 0} \frac{1}{\Lambda_k} < \infty$

Similar to Equation (4.11), we can derive truncation points for the sum above from the probability distribution of $X^B(t)$, that is, for $\epsilon > 0$, we choose truncation points L and R such that

$$\sum_{k=L}^{R} \mathcal{B}_{\mathbf{\Lambda}_{(k)} \cdot t}(k) \geq 1 - \epsilon.$$

Since the sets $\mathcal{S}_k$ are constructed during the iteration, the values $\Lambda_0, \Lambda_1, \ldots$ are not known a-priori, and nor are the truncation points L and R. We can, however, set $L = 0$ and add up summands $\boldsymbol{w}^{(k)} \cdot \mathcal{B}_{\mathbf{\Lambda}_{(k)} \cdot t}(k)$ until the entries of the current approximation of $\boldsymbol{g}^{(t)}$ sum up to at least $1 - \epsilon$.

If $\sup_{\mathbf{s} \in \mathcal{S}} \lambda_s = \Lambda < \infty$, we can compare Equation (4.17) and Equation (4.8). We observe that $\Lambda_k \leq \Lambda$ for all k. Hence, for any infinitesimal time interval $[h, h + dt)$,

$$\begin{aligned} & Pr\big(X^B(h + dt) = k + 1 \,|\, X^B(h) = k\big) = \Lambda_k \cdot dt \\ \leq \quad & Pr\big(X^P(h + dt) = k + 1 \,|\, X^P(h) = k\big) = \Lambda \cdot dt. \end{aligned}$$

This means that during the interval $[0, t)$, the Poisson process $X^P(t)$ has at least as many jumps as $X^B(t)$. This implies that the truncation of the sum in Equation (4.17) w.r.t. a given accuracy ϵ may yield a smaller right truncation point compared to the truncation in Equation (4.11). Hence, fewer matrix-vector multiplications have to be carried out. If the computational complexity of the algorithm is dominated by the computation of the vectors $\boldsymbol{w}^{(k)}$, AU outperforms SU. For a large time horizon t, however, the right truncation point for AU often approaches that of SU, i.e., after a certain number ℓ of steps, $\Lambda_k = \Lambda$ for all $k \geq \ell$. Then AU becomes less efficient than SU.

Further drawbacks of adaptive uniformization are the fact that the computation of the values $Pr\big(X^B(t) = k\big)$ is more costly than the computation of the values $Pr\big(X^P(t) = k\big)$. The reason is that closed form expressions for the solution of a birth process do not give rise, in general, to numerically stable algorithms. Moreover, as mentioned above, the truncation points cannot be calculated a-priori since the construction of X^B is part of the iterative computation of the vectors $\boldsymbol{w}^{(1)}, \boldsymbol{w}^{(2)}, \ldots$.

Solution of the birth process. For the computation of the probabilities $\mathcal{B}_{\mathbf{\Lambda}_{(k)} \cdot t}(k) = Pr\big(X^B(t) = k\big)$, we construct a propagation model $N^B = \big\langle \mathbb{N}_0, [0, 1], \zeta^B, \pi^B \big\rangle$, with initial mass vector $\zeta^B(0) = 1$, and with edge function π^B determined by a generator matrix Q^B. The generator matrix is a simple

infinite matrix Q^B with entries are $Q^B(k,k) = -\Lambda_k$, $Q^B(k,k+1) = \Lambda_k$, for $k \in \{0,1\ldots\}$, and zero elsewhere. We use standard uniformization to solve the continuous-time problem defined by N^B, and obtain a subordinated propagation model $N^{B,u}$ with transition probability matrix $P^B = I + \frac{1}{\Lambda}Q^B$, where $\Lambda \geq \sup_{k\geq 0}\Lambda_k$. For ease of presentation, let $\boldsymbol{b}$ denote the discrete-time propagation process $\boldsymbol{f}^{B,d}$ of propagation model $N^{B,u}$. Thus,

$$\begin{aligned}\mathcal{B}_{\boldsymbol{\Lambda}_{(k)}\cdot t}(k) &= \sum_{l=0}^{\infty} b_k^{(l)} \cdot \mathcal{P}_{\Lambda\cdot t}(l),\\ &\approx \sum_{l=L'}^{R'} b_k^{(l)} \cdot \mathcal{P}_{\Lambda\cdot t}(l). \end{aligned} \tag{4.18}$$

The matrix P^B inherits the simple structure of Q^B. The entry $P^B(k,k+1)$ equals $\frac{\Lambda_k}{\Lambda} =: a_k$, and the diagonal entry $P^B(k,k) = 1 - a_k$. Then

$$b_k^{(l)} = a_{k-1}\cdot b_{k-1}^{(l-1)} + (1-a_k)\cdot b_k^{(l-1)}, \tag{4.19}$$

that is, after l steps, the birth process is in state k if after $l-1$ steps it is in the state $k-1$ and takes a transition to state k, or after $l-1$ steps it is in state k and takes the self-loop. In order to compute $b_k^{(l)}$, we only need the transition rates $\Lambda_0,\ldots,\Lambda_k$ but not $\Lambda_{k+1},\Lambda_{k+2},\ldots$. It is important to point out that for the birth process we can afford the large number of iterations that are necessary during SU. The reason is that the simple structure of $N^{B,u}$ permits a fast computation of the values $b_k^{(l)}$. Moreover, similar to our strategy for the solution of $\boldsymbol{w}$, we set entries in $\boldsymbol{b}^{(l)}$ to zero if they drop below the threshold δ. This introduces an additional approximation error for $b_k^{(l)}$, but results in a significant speed-up.

If we combine Equation (4.18) and the solution of the DTPP $\mathbf{w}$, we obtain an approximation $\hat{\boldsymbol{g}}^{(t)}$ for $\boldsymbol{g}^{(t)}$, that is,

$$\begin{aligned}\boldsymbol{g}^{(t)} &\approx \sum_{k=0}^{R} \boldsymbol{w}^{(k)} \cdot Pr\big(X^B(t) = k\big)\\ &\approx \sum_{k=0}^{R} \boldsymbol{w}^{(k)} \cdot \sum_{l=L'}^{R'} b_k^{(l)} \cdot \mathcal{P}_{\Lambda\cdot t}(l) =: \hat{\boldsymbol{g}}^{(t)}. \end{aligned} \tag{4.20}$$

The outer sum is only truncated on the right and the truncation point R is found during the AU-iteration. For the inner sum, we can compute L' and R' a-priori as mentioned above for SU. Due to the simple structure of X^B, however,

it is possible to derive closer truncation points. Instead of deriving L' and R' only from the inequality

$$\sum_{l=L'}^{R'} \mathcal{P}_{\Lambda \cdot t}(l) > 1 - \epsilon,$$

we choose dynamical truncation points L'_k and R'_k depending on the probabilities $b_k^{(l)}$. More precisely, we choose $[L'_k, R'_k]$ to be the smallest interval that includes all integers l for which $b_k^{(l)} > \delta$.

Both truncations in Equation (4.20) lead to an underapproximation of the true value. The same holds for the error introduced by neglecting states in N^a and $N^{B,u}$ whose entries in $\boldsymbol{w}^{(k)}$ and $\boldsymbol{b}^{(l)}$ drop below a certain threshold, respectively. Thus, the approximation $\hat{\boldsymbol{g}}^{(t)}$ is an underapproximation of $\boldsymbol{g}^{(t)}$ and the total error is given by $1 - \sum_{\mathbf{s} \in \mathcal{S}} \hat{g}_{\mathbf{s}}^{(t)}$. In our experimental results, we report the total error using different values for δ. In the case that an a-priori specified error bound has to be met it is possible to repeat steps of the iteration if the total error exceeds the specified bound.

AU for non-linear propagation models. So far we have seen how to solve the CTPP problem through adaptive uniformization for linear propagation models. As in the case of standard uniformization, we will now give an adaptive uniformization method for propagation models for which the approximation (4.12) is acceptable.

Theorem 4.2. *For a propagation model $N = \langle \mathcal{S}, \mathcal{M}, \zeta, \pi \rangle$, let N^a be a second PM, with $N^a = \langle \mathcal{S}, \mathbb{N}_0 \times \mathcal{M}, \zeta^a, \pi^a \rangle$, where $\zeta^a = \langle \mathbf{0}, \zeta \rangle$ and*

$$\pi^a_{\mathbf{s} \to \mathbf{s}'}(\langle k, \Lambda \rangle) = \left\langle 1, \frac{1}{\Lambda_k} \pi_{\mathbf{s} \to \mathbf{s}'}(\Lambda) \right\rangle,$$

where $\Lambda_k = max_{\mathbf{s}: w_{\mathbf{s}}^{(k)} > 0} \lambda_s$.

Recall that f^a is the discrete time propagation process of N^a. Then, if Equation 4.12 is exact we have that:

$$g_{\mathbf{s}'}^{(t)} = \sum_{k=0}^{\infty} \mathcal{B}_{(\boldsymbol{\Lambda}_{(k)} \cdot t)}(k) \cdot f_{2,\mathbf{s}'}^{a(k)},$$

where $\boldsymbol{f}_2^{a(k)}$ is the second component of the propagation process $\boldsymbol{f}$.

Proof. Let $\Lambda = \sup_k \Lambda_k$, and recall that $\mathbf{\Lambda}_{(k)}$ denotes a series with elements the values Λ_k. By using standard uniformization, we have that:

$$\mathcal{B}_{(\mathbf{\Lambda}_{(k)} \cdot t)}(k) = \sum_{k'=0}^{\infty} \mathcal{P}_{(\Lambda \cdot t)}(k') b_k^{(k')},$$

where the vectors $\boldsymbol{b}^{(k')}$ are the solutions of a discrete-time propagation problem, and have as elements the values $b_k^{(k')}$. Furthermore:

$$\begin{aligned}
\frac{d\mathcal{B}_{(\mathbf{\Lambda}_{(k)} \cdot t)}(k)}{dt} &= \Lambda \sum_{k'=0}^{\infty} (\mathcal{P}_{\Lambda \cdot t}(k'-1) - \mathcal{P}_{\Lambda \cdot t}(k')) \cdot b_k^{(k')} \\
&= \Lambda \sum_{k'=0}^{\infty} \mathcal{P}_{\Lambda \cdot t}(k') (b_k^{(k'+1)} - b_k^{(k')}) \\
&= \Lambda \sum_{k'=0}^{\infty} \mathcal{P}_{\Lambda \cdot t}(k') \left(-b_k^{(k')} \cdot \frac{\Lambda_k}{\Lambda} + b_{k-1}^{(k')} \frac{\Lambda_{k-1}}{\Lambda} \right) \\
&= \sum_{k'=0}^{\infty} \mathcal{P}_{\Lambda \cdot t}(k') \left(b^{k'}(k-1) \Lambda_{k-1} - b_k^{(k')} \cdot \Lambda_k \right)
\end{aligned}$$

We will show that $\sum_{k=0}^{\infty} \mathcal{B}_{(\mathbf{\Lambda}_{(k)}\cdot t)}(k) \cdot f_{2,\mathbf{s}'}^{a(k)}$ verifies equation(2.9). To do so, we compute its derivative:

$$
\begin{aligned}
&\frac{d\sum_{k=0}^{\infty} \mathcal{B}_{(\mathbf{\Lambda}_{(k)}\cdot t)}(k) \cdot f_{2,\mathbf{s}'}^{a(k)}}{dt} \\
=&\sum_{k=0}^{\infty} \frac{d\mathcal{B}_{(\mathbf{\Lambda}_{(k)}\cdot t)}(k)}{dt} \cdot f_{2,\mathbf{s}'}^{a(k)} \\
=&\sum_{k=0}^{\infty}\sum_{k'=0}^{\infty} \mathcal{P}_{\Lambda\cdot t}^{(k')} \cdot (b^{k'}(k-1)\Lambda_{k-1} - b_k^{(k')} \cdot \Lambda_k) \cdot f_{2,\mathbf{s}'}^{a(k)} \\
=&\sum_{k=0}^{\infty} \Lambda_k \cdot \sum_{k'=0}^{\infty} \mathcal{P}_{\Lambda\cdot t}^{(k')} \cdot b_k^{(k')} \cdot (f_{2,\mathbf{s}'}^{d(k+1)} - f_{2,\mathbf{s}'}^{a(k)}) \\
=&\sum_{k=0}^{\infty} \Lambda_k \cdot \mathcal{B}_{\Lambda_k,t}(k) \cdot (f_{2,\mathbf{s}'}^{d(k+1)} - f_{2,\mathbf{s}'}^{a(k)}) \\
=&\sum_{k=0}^{\infty} \Lambda_k \cdot \mathcal{B}_{\Lambda_k,t}(k) \cdot \left(\sum_{\mathbf{s}\in\mathcal{S}} \pi_{\mathbf{s}\to\mathbf{s}'}^{a}(f_{2,\mathbf{s}}^{a(k)}) - \sum_{\mathbf{s}''\in\mathcal{S}} \pi_{\mathbf{s}'\to\mathbf{s}''}^{a}(f_{2,\mathbf{s}'}^{a(k)}) + \pi_{\mathbf{s}'\to\mathbf{s}'}^{a}(f_{2,\mathbf{s}'}^{a(k)})\right) \\
=&\sum_{\mathbf{s}\in\mathcal{S}} \Lambda_k \cdot \pi_{P,\mathbf{s}\to\mathbf{s}'}(\sum_{k=0}^{\infty} \mathcal{B}_{\Lambda_k,t}(k) \cdot f_{2,\mathbf{s}}^{a(k)}) \\
-&\sum_{\mathbf{s}''\in\mathcal{S}} \Lambda_k \cdot \pi_{P,\mathbf{s}'\to\mathbf{s}''}(\sum_{k=0}^{\infty} \mathcal{B}_{\Lambda_k,t}(k) \cdot f_{2,\mathbf{s}'}^{a(k)}) \\
+&\Lambda_k \cdot \pi_{P,\mathbf{s}'\to\mathbf{s}'}(\sum_{k=0}^{\infty} \mathcal{B}_{\Lambda_k,t}(k) \cdot f_{2,\mathbf{s}'}^{a(k)}) \\
=&\sum_{\mathbf{s}\in\mathcal{S}} \pi_{\mathbf{s}\to\mathbf{s}'}(g_{\mathbf{s}}^{(t)}) - \sum_{\mathbf{s}''\in\mathcal{S}} \pi_{\mathbf{s}'\to\mathbf{s}''}(g_{\mathbf{s}'}^{(t)}) + \pi_{\mathbf{s}'\to\mathbf{s}'}(g_{\mathbf{s}'}^{(t)}) \\
=&\frac{dg_{\mathbf{s}'}^{(t)}}{dt}.
\end{aligned}
$$

□

4.3.3 Fast Adaptive Uniformization (FAU)

By combining adaptive uniformization with the on-the-fly space construction and with the threshold abstraction that we introduced in Chapter 3, we derive a variant of adaptive uniformization that we call *fast adaptive uniformization* which computes the under-approximations $\hat{\mathbf{p}}^{(t_1)}, \hat{\mathbf{p}}^{(t_1)}, \ldots$.

This brings several important advantages that we list below.

– *Smaller Right Truncation Point:* The sets $\mathcal{S}_0, \mathcal{S}_1, \ldots$ may contain fewer states because the definition of $\mathcal{S}_k$ depends on $\boldsymbol{w}^{(k)}$. Thus, $\Lambda_0, \Lambda_1, \ldots$ are replaced by the possibly smaller $\hat{\Lambda}_0, \hat{\Lambda}_1, \ldots$ since they are the maximal exit rates over fewer

	fast AU				fast SU		
thres. δ	**ex. time**	**states**	**error**	**iterat.**	**ex. time**	**states**	**error**
e−11	62s	1.3e4	9e−6	455240	1718s	7.3e3	4e−2
e−12	62s	1.4e4	1e−6	455250	2129s	9.1e3	4e−3
e−13	63s	1.4e4	2e−7	455259	2460s	1e4	4e−4
e−14	64s	1.7e4	1e−7	455267	2759s	1.2e4	3e−5
e−15	67s	1.7×10^4	1e−7	455274	3001s	1.4e4	4e−6
0	1161s	4.8e4	1e−7	457242	TO 3h	-	-

Table 4.4: Results for the crystallization example.

	fast AU				fast SU		
thres. δ	**ex. time**	**states**	**error**	**iterat.**	**ex. time**	**states**	**error**
e−11	512s	2e5	7e−3	2e4	1450s	1e5	3e−2
e−12	930s	3e5	1e−3	2.3e4	2675s	2e5	4e−3
e−13	1502s	4e5	1e−4	2.6e4	4229s	3e5	6e−4
e−14	2286s	5e5	2e−5	2.9e4	6122s	4e5	7e−5
e−15	3326s	7e5	3e−6	3.2×10^4	8434s	6e5	1e−5
0	TO 3h	-	-	-	TO 3h	-	-

Table 4.5: Results for the phage λ example.

states. In this case, the birth process X^B has smaller transition probabilities than the birth process used in the original AU algorithm. In the case of smaller transition probabilities, less probability mass moves rightwards within $[0, t)$ and thus the right truncation point is smaller.

– *Smaller Vectors:* Each vector-matrix multiplication $\boldsymbol{w}^{(k)} \cdot P_k$ requires less computational effort since $\boldsymbol{w}^{(k)}$ contains fewer nonzero entries.

– *Non-explosive Birth Process:* For infinite propagation models, the threshold abstraction also ensures that the limit of the sequence $\mathcal{S}_0, \mathcal{S}_1, \ldots$ will remain finite since only finitely many states can have a probability greater δ. Therefore, the sequence $\Lambda_0, \Lambda_1, \ldots$ will be bounded even if $\sup_{\mathbf{s}\in\mathcal{S}} \lambda_s = \infty$. Thus, X^B does not explode.

4.3.4 Case Studies

We implemented both methods fast AU (FAU) and fast SU (FSU) in C++ and run experiments on a 3.16 GHz Intel Linux PC with 6 GB of RAM. We consider

three examples, of which our most complex example has 12 different chemical species and 19 reactions. We approximate the full probability distribution at a single point in time. Note that FAU and FSU can also be used in an iterative fashion to compute the mass distribution of a PM at several time instances. For instance, it is possible to iteratively calculate the mass distribution after equidistant time intervals by using the distribution of the previous step as initial distribution of the current step. All of our examples come from biological systems and require the transient solution of a Markov chain, which is given by the chemical master equation.

Example 4.4 (**Simple Crystallization**). *We consider a simple crystallization example that involves the chemical species A, B, C, and D. The two possible reaction types are given by $2A \to B$ and $A + C \to D$. Hence $\mathcal{S} \subseteq \mathbb{N}^4$ and for a state $(x_1, x_2, x_3, x_4) \in \mathcal{S}$ the propensity functions are $\alpha_1(x_1, x_2, x_3, x_4) = c_1\binom{x_1}{2} = c_1 x_1(x_1-1)/2$ and $\alpha_2(x_1, x_2, x_3, x_4) = c_2 x_1 x_3$, where c_1, c_2 are positive constants.*

The first example is the simple crystallization in Example 4.4. The underlying CTMC is finite due to the conservation of mass in the model. We chose rate constants $c_1 = c_2 = 10^{-7}$ and initial state $\mathbf{y} = (10^6, 0, 10, 0)$ [50]. We present our experimental results for the approximation of the probability distribution at time $t = 100$ in Table 4.4. The columns 2-5 of the table contain the results of FAU and the columns 6-8 those of FSU. Each row corresponds to a different choice of the threshold δ (listed in the first column), that is, we use different values for the threshold abstraction of the DTMC X^u or X^a. For both methods we list the execution time (**ex. time**), the average size of the set S_k (**states**), and the total error (**error**) $1 - \sum_{\mathbf{s}\in\mathcal{S}} \hat{p}_{\mathbf{s}}^{(t)}$. For FAU, we also list the total number of iterations (**iterat.**), i.e. the number of steps for which we compute the solution of the DTMC. In contrast, the number of iterations required by SU is 5011731 (for all choices of δ).

The last row shows results for the case that $\delta = 0$, that is, no threshold abstraction was used for the DTMC. In this case the number of states with a positive entry in the vector $\hat{w}^{(k)}$ becomes intractably large. The entry "TO" refers to the case where the execution time exceeded the time out interval of 3 hours.

For the crystallization example, FAU performs significantly better than FSU. The reason is that the adaptive rates $\hat{\Lambda}_0, \hat{\Lambda}_1, \ldots$ decrease as the number of molecules of type A becomes smaller. As opposed to that FSU uses a birth

$L + R \xrightarrow{c_1} B_0$	$c_1 = 6.7\text{e}{-3}$	receptor-ligand binding
$B_{j+1} \xrightarrow{c_j} B_{j+2}$	$c_j = 0.25$	forward modifications for $j \in \{2, \ldots, 7\}$
$B_{j-5} \xrightarrow{c_j} L + R$	$c_j = 0.5$	backward modifications for $j \in \{8, \ldots, 14\}$
$B_6 + X \xrightarrow{c_{15}} C$	$c_{15} = 1.2\text{e}{-3}$	binding of inactive mess.
$C \xrightarrow{c_{16}} B_6 + X$	$c_{16} = 1\text{e}{-2}$	unbinding of inactive mess.
$C \xrightarrow{c_{17}} B_6 + X'$	$c_{17} = 100$	release of activated mess.
$C \xrightarrow{c_{18}} L + R + X$	$c_{18} = 0.5$	unbinding of inactive mess. and ligands
$X' \xrightarrow{c_{19}} X$	$c_{19} = 2\text{e}{-3}$	inactivation of messengers

Table 4.6: Reactions of the signaling example.

	fast AU				**fast SU**		
thres. δ	**ex. time**	**states**	**error**	**iterat.**	**ex. time**	**states**	**error**
e−11	$45s$	1e5	4e−4	970	$49s$	8e4	7e−4
e−12	$134s$	2e5	1e−4	1105	$149s$	2e5	1e−4
e−13	$325s$	5e5	2e−5	1202	$370s$	4e5	4e−5
e−14	$697s$	1e6	5e−6	1276	$796s$	8e5	8e−6
e−15	$1402s$	1e6	1e−6	1350	$1557s$	1e6	1e−6
0	o.o.m.	-	-	-	o.o.m.	-	-

Table 4.7: Results of the signaling example.

process that jumps at constant rate $\hat{\Lambda} = \max_{k \leq R'} \hat{\Lambda}_k$ which results in very short time steps even when the dynamics of the system slows down.

Goutsias' Model As a second example we reconsider Example 4.2, a model for the transcription regulation of a repressor protein in bacteriophage λ [44]. This protein is responsible for maintaining lysogeny of the λ virus in E. coli [5]. The model involves 6 different species and 10 reactions. Thus, a state is a vector $x \in \mathbb{N}_0^6$, where the i-th entry refers to the number of molecules of type M, D, RNA, DNA, $DNA.D$ and $DNA.2D$, respectively. Note that infinitely many states are reachable in the corresponding CTMC. The initial state of the system is given by $\mathbf{y} = (2, 6, 0, 2, 0, 0)$ and the time horizon is $t = 300$. We present our experimental results in Table 4.5. Note that for the crystallization example the total error is similar for all choices of δ whereas for the phage λ example the total error decreases for smaller δ. By comparing the column

iterat. with the number of iterations performed by SU, 1.5×10^5, we conclude that AU requires significantly less DTMC steps than SU. In the case of $\delta = 0$, the system times out after three hours as the space size is too big to handle.

Similar to the crystallization example, the fast AU method performs significantly better than the fast SU method. Again, the reason is that the dynamics of the system changes. This example has also been solved approximately by Sidje et al. [12].

Immune-Receptor Signaling We consider a model of intracellular signaling through immune receptors that are involved in antigen recognition [42]. The model consists of 12 different chemical species and 19 reactions. After binding to a receptor, a ligand undergoes six modifications and can generate a signal by activating a messenger. In [42] this model is analyzed using Monte-Carlo simulation. We list the reactions in Table 4.6.

Following [42], the rate constants are chosen as shown in column 2 of Table 4.6 and the initial state is $x = (x_1, \ldots, x_{12})$ with $x_1 = 30$ ligands, $x_2 = 900$ receptors and $x_{10} = 10000$ messengers. We compute the probability distribution of the underlying CTMC at time $t = 4$. In Table 4.7, we list our experimental results. As opposed to the previous examples, FSU performs almost as good as FAU for the immune-receptor signaling example. The reason is that in this system the dynamics do not change significantly, i.e. the values $\hat{\Lambda}_0, \hat{\Lambda}_1, \ldots$ are not considerably different from the global uniformization rate used for fast SU. Thus, the number of iterations needed by FSU, 2056, is of the same order of magnitude as the number of iterations need by fast AU (see column **iterat.**). If $\delta = 0$ the memory requirements become enormous and after few iterations the system is out of memory (we indicate this by "o.o.m." in Table 4.7).

4.4 Runge-Kutta Method

In this section we apply an ODE solver to reduce a continuous-time propagation problem to the discrete-time case. More specifically, the algorithm that we propose here is based on the numerical integration of Equation (2.9) using an explicit fourth-order Runge-Kutta method (RK4).

Let N be a propagation model defined by the tuple $\langle \mathcal{S}, \mathcal{M}, \zeta, \pi \rangle$ whose continuous-time propagation process g follows the differential equation:

$$\frac{dg^{(t)}_{\mathbf{s}'}}{dt} = \sum_{\mathbf{s} \in \mathcal{S}} \pi_{\mathbf{s} \to \mathbf{s}'}(g^{(t)}_{\mathbf{s}}) - \sum_{\mathbf{s}'' \in \mathcal{S}} \pi_{\mathbf{s}' \to \mathbf{s}''}(g^{(t)}_{\mathbf{s}'}) + \pi_{\mathbf{s}' \to \mathbf{s}'}(g^{(t)}_{\mathbf{s}'}),$$
$$\text{with initial condition: } \boldsymbol{g}^{(0)} = \zeta.$$

The standard explicit fourth-order Runge-Kutta method applied this equation yields the iteration step [105]

$$\boldsymbol{g}^{(t+h)} = \boldsymbol{g}^{(t)} + h \cdot (\boldsymbol{k}_1 + 2 \cdot \boldsymbol{k}_2 + 2 \cdot \boldsymbol{k}_3 + \boldsymbol{k}_4)/6, \tag{4.21}$$

where $h > 0$ is the time step of the method and the vectors $\boldsymbol{k}_1, \boldsymbol{k}_2, \boldsymbol{k}_3, \boldsymbol{k}_4$ are given by

$$\begin{aligned}
k_1(\mathbf{s}') &= \pi_{\mathbf{s}' \to \mathbf{s}'}(g^{(t)}(\mathbf{s}')) \\
&\quad + \sum_{\mathbf{s} \in \mathcal{S}} \big(\pi_{\mathbf{s} \to \mathbf{s}'}(g^{(t)}_{\mathbf{s}}) \\
&\quad - \pi_{\mathbf{s}' \to \mathbf{s}}(g^{(t)}(\mathbf{s}'))\big) \\
k_{l+1}(\mathbf{s}') &= \pi_{\mathbf{s}' \to \mathbf{s}'}(g^{(t)}(\mathbf{s}') + h \cdot k_l(\mathbf{s}')/2)) \\
&\quad + \sum_{\mathbf{s} \in \mathcal{S}} \big(\pi_{\mathbf{s} \to \mathbf{s}'}(g^{(t)}_{\mathbf{s}} + h \cdot k_l(\mathbf{s})/2)) \\
&\quad - \pi_{\mathbf{s}' \to \mathbf{s}}(g^{(t)}(\mathbf{s}') + h \cdot k_l(\mathbf{s}')/2))\big), \text{ for } l \in \{1, 2\}, \\
k_4(\mathbf{s}') &= \pi_{\mathbf{s}' \to \mathbf{s}'}(g^{(t)}(\mathbf{s}') + h \cdot k_3(\mathbf{s}')) \\
&\quad + \sum_{\mathbf{s} \in \mathcal{S}} \big(\pi_{\mathbf{s} \to \mathbf{s}'}(g^{(t)}_{\mathbf{s}} + h \cdot k_3(\mathbf{s})) \\
&\quad - \pi_{\mathbf{s}' \to \mathbf{s}}(g^{(t)}(\mathbf{s}') + h \cdot k_3(\mathbf{s}'))\big).
\end{aligned} \tag{4.22}$$

We construct a second propagation model $N^R = \langle \mathcal{S}, \mathcal{M}^R, \zeta^R, \pi^R \rangle$ such that:

$$\boldsymbol{g}^{(n \cdot h)} \approx \boldsymbol{f}^{R(5 \cdot n)} \cdot \mathbf{1}_m,$$

where $\mathbf{1}_m$ is a column vector with entry 1 at position 2 and 0 everywhere else.

The mass space of N^R is $\mathcal{M}^R = \mathbb{N}_0 \times \mathcal{M} \times \mathcal{M}^4$. The first field corresponds to the step number, the second to the N-mass of the state (which represents the mass of the PM N), and the last 4 to the fields k_l.

The initialization function of N^R is:

$$\zeta^R_{\mathbf{s}} = (0, \zeta_{\mathbf{s}}, \mathbf{0}).$$

We define the edge function of N^R by its components: $\pi^R = (\pi^k, \pi^\mu, \pi^{k_l})$.

$$\pi^n_{\mathbf{s}\to\mathbf{s}}(\langle n, \mu, \boldsymbol{k}\rangle) = 1,$$

$$\pi^{k_1}_{\mathbf{s}\to\mathbf{s}'}(\langle 5\cdot n, \mu, \boldsymbol{k}\rangle) = \pi_{\mathbf{s}\to\mathbf{s}'}(\mu),$$

$$\pi^{k_2}_{\mathbf{s}\to\mathbf{s}'}(\langle 5\cdot n+1, \mu, \boldsymbol{k}\rangle) = \pi_{\mathbf{s}\to\mathbf{s}'}(\mu + \frac{h}{2}\cdot k_1),$$

$$\pi^{k_3}_{\mathbf{s}\to\mathbf{s}'}(\langle 5\cdot n+2, \mu, \boldsymbol{k}\rangle) = \pi_{\mathbf{s}\to\mathbf{s}'}(\mu + \frac{h}{2}\cdot k_2),$$

$$\pi^{k_4}_{\mathbf{s}\to\mathbf{s}'}(\langle 5\cdot n+3, \mu, \boldsymbol{k}\rangle) = \pi_{\mathbf{s}\to\mathbf{s}'}(\mu + h\cdot k_3),$$

$$\pi^{\mu}_{\mathbf{s}\to\mathbf{s}}(\langle 5\cdot n+4, \mu, \boldsymbol{k}\rangle) = \frac{h}{6}\cdot(k_1 + 2\cdot k_2 + 2\cdot k_3 + k_4),$$

$$\pi^{k_r}_{\mathbf{s}\to\mathbf{s}}(\langle 5\cdot n+4, \mu, \boldsymbol{k}\rangle) = -k_r,$$

$$\pi^R_{\mathbf{s}\to\mathbf{s}'}(\mu) = 0, \text{if not already defined above.}$$

4.4.1 Algorithm

In order to solve a continuous-time propagation problem, we apply the ideas presented in Chapter 3 to the propagation model Ns. The main idea is to integrate only those differential equations of the PME (see Equation (2.9)) that correspond to states with "significant mass". This reduces the computational effort significantly since in each iteration step only a comparatively small subset of states is considered. We dynamically decide which states to drop/add based on a fixed mass threshold $\delta > 0$. In the case of propagation models that are derived from biochemical reaction netoworks, due to the regular structure of the corresponding Markov model the approximation error of the algorithm remains small since probability mass is usually concentrated at certain parts of the state space. The farther away a state is from a "significant set" the smaller is its probability. Thus, the total error of the approximation remains small. Unless otherwise specified, in our experiments we fix δ to 10^{-14}, which has been shown to lead to accurate approximations [23]. Since in each iteration step some mass is "lost" we obtain a substochastic probability vector and the approximation error is the sum of all probability mass lost (provided that the numerical integration could be performed without any errors).

In order to avoid the explicit construction of $\mathcal{S}$ we work with a dynamic set $\mathbf{S}$ of significant states that changes in each step, we use for a state $\mathbf{s}$ a data structure x with the following components:

Algorithm 4.6 A single iteration step of the fast RK4 algorithm, which approximates the solution of the PME.

```
 1: for l = 1, 2, 3, 4 do  // traverse S four times
 2:   switch l  // decide which fields from state data structure are needed for k_l
 3:     case l = 1: coeff := 1; field := μ;
 4:     case l ∈ {2, 3}: coeff := h/2; field := k_{l−1};
 5:     case l = 4: coeff := h; field := k_{l−1};
 6:   for all s ∈ S do
 7:     x.k_l ← x.k_0;  // small speed up
 8:     for π_{x.s→s'}(x.μ) > 0 do
 9:       x' = find(s', S);
10:       if x' = null then
11:         S := S ∪ {x'};
12:       x.k_l := x.k_l − coeff · π_{x.s→s'}(x.field);
13:       x'.k_l := x'.k_l + coeff · π_{x.s→s'}(x.field);
14:     x.k_l := x.k_l + coeff · π_{x.s→x.s}(x.field);  // for non-conservative PM
15: for all s ∈ S do
16:   x.μ := x.μ + h · (x.k_1 + 2 · x.k_2 + 2 · x.k_3 + x.k_4)/6;
17:   x.k_1 := 0; x.k_2 := 0; x.k_3 := 0; x.k_4 := 0;
18:   if x.μ < δ then
19:     S := S \ {x};
```

– a field $x.\mu$ for the current N-mass of $\mathbf{s}$,
– fields $x.k_1, \ldots, x.k_4$ for the four terms in the equation of state $\mathbf{s}$ in the system of Equation (4.22),
– a pointer to the discovered successors $\mathbf{s}'$ of $x.\mathbf{s}$.

We start at time $t = 0$ and initialize the set $\mathbf{S}$ as the set of all states that have initially a mass greater than $\delta \in \mathcal{M}$, l.e. $\mathbf{S}$ starts with nodes for the set of states $\left\{\mathbf{s} \mid g_{\mathbf{s}}^{(0)} > \delta\right\}$. We perform a step of the iteration in Equation (4.21) by traversing the set $\mathbf{S}$ five times. In the first four rounds we compute $\boldsymbol{k}_1, \ldots, \boldsymbol{k}_4$ and in the final round we accumulate the summands. While processing node x in round l, $l < 5$, we propagate mass to each successor of $x.s$ ($\pi_{x.\mathbf{s}\to x'.s}(x.\mu) > 0$), by subtracting a term from $x.k_l$ (see Equation (4.22)) and adding to the same field k_l of x'. A single iteration step is illustrated in pseudocode in Table 4.6. In line 18, we ensure that $\mathbf{S}$ does not contain states with a mass value less than δ. In lines 2-14 we compute the values $k_{1,\mathbf{s}}, \ldots, k_{4,\mathbf{s}}$ for all $x \in \mathbf{S}$ (see Equation (4.22)). The fifth round starts in line 15 and in line 16 the approximation of the mass $g_{\mathbf{s}}^{(t+h)}$ is calculated. Note that the fields $x.k_1, \ldots, x.k_4$ are reset to with zero.

Chapter 5

Related Work

Here, we thoroughly relate our work with the alternative for numerical solutions: stochastic simulation, with the alternative for threshold abstraction or sliding window abstraction: finite state projection, and finally with the alternative for uniformization and Runge-Kutta method: Krylov sub-space methods. But first we give an overview of all methods that related to sliding windows and to fast adaptive uniformization.

Various abstraction techniques for Markov chains with *finite* state spaces have been developed during the last years [19, 20, 66, 71]. Infinite-state Markov chains with *discrete time* have been considered in the context of probabilistic lossy-channel systems [1–3, 91] and probabilistic pushdown systems [29–31, 68]. In the infinite-state continuous-time setting, model-checking algorithms for quasi-birth-death processes and Jackson queuing networks have been studied by Remke [95], where the underlying Markov chains are highly structured and represent special cases of CTMCs defined by transition classes. The closest work to the *sliding window method* is the model-checking algorithm for infinite-state CTMCs by Zhang et al. [112]. Depending on the desired precision, their algorithm simply explores the reachable states up to a finite path depth. In contrast, our approach takes into account the direction into which the probability mass moves, and constructs a sequence of abstract models "on-the-fly," during the verification process. Similar approaches have also been used in the context of biochemical reaction networks. Similar to [112], Munsky et al. [83] explore models up to a specified finite path depth, whereas Burrage et al. [12] consider a finite projection that is doubled if necessary. The latter method, however, requires a priori knowledge about the direction and spread of the probability mass.

The Fokker-Planck equation is an approximation of the CME, for which a solution can be obtained efficiently [103, 104]. This approximation, however, does not take into account the discrete nature of the system, but changes the underlying model by assuming a continuous state space. Other approaches to approximate the probability distributions defined by the CME are based on sparse grid methods [51], spectral methods [28], or the separation of time scales [13, 88]. The latter approach uses a quasi-steady state assumption for a subset of chemical species and calculates the solution of an abstract model of the system. In contrast, we present an algorithm that computes a direct solution of the CME. Our method is also related to tau-leaping techniques [14, 40], because they require estimates of the upper and lower bounds on the population sizes of the chemical species, just as our method. The *time leap* must be sufficiently small such that the changes in the population vector do not significantly affect the dynamics of the system. Our method differs from the calculation of the leap in predicting the future dynamics for a dynamically chosen time period. More precisely, we determine the length of the next time step while approximating the future behavior of the process.

Recently, uniformization has been used in the context of biochemical reaction networks. Hellander [53] combines SU with Monte Carlo simulation. Sidje et al. also consider SU and, similar to our approach, they neglect states in the DTMC that have insignificant probability [100]. Zhang et al. apply the external uniformization method to biochemical reaction networks [111]. As opposed to the approaches mentioned above *fast adaptive uniformization* modifies both, the solution approach for the DTMC as well as the solution approach for the associated birth process.

5.1 Stochastic Simulation

Two different families of computational approaches have been proposed and used to estimate event probabilities and approximate probability distributions. The first kind of approach is based on numerical simulation, i.e., the generation of many sample trajectories (or *simulation runs*) of the system, and the second kind is the subject of this thesis, numerical analysis. The former approach is known as *Gillespie simulation* [38], in which pseudo-random numbers are used to simulate molecular noise. Measures of interest are obtained via statistical output analysis. The main advantage of simulation is that it is easy to implement and the generation of trajectories is not limited by the size of the state space. Moreover, the precision level of the method can be easily adjusted by

performing more or fewer simulation runs. For the computation of the probability of certain events, however, simulative approaches become computationally expensive, because a large number of runs have to be carried out to bound the statistical error appropriately. For estimating event probabilities, a higher precision level is necessary than for estimating cumulative measures such as expectations, and simulation becomes expensive because doubling the precision requires four times more simulation runs.

In contrast, approaches based on a numerical reachability analysis approximate probability distributions of the CTMC. As opposed to a statistical estimation of probabilities, which yields an indirect solution, the master equation is numerically solved by integrating the system's behavior over time. Standard numerical techniques are impractical for many systems because of the well-known problem of state space explosion. Recently, however, more sophisticated numerical approximation methods have been proposed, which solve the system in an iterative fashion and consider only subsets of the state space during any given time interval, such as our own work presented here, and finite state projection methods[12, 83]. They are significantly more efficient than global analysis because they use localization optimizations (such as "sliding windows") and dynamic adaptation ("on-the-fly" generation of windows). These methods efficiently compute the probability distribution of large CTMC at several time instances up to a small approximation error. They can also be used for infinite-state systems.

In this section, we evaluate and compare the performance of the two different approaches for the computation of probabilities of certain events, i.e., the statistical estimation using simulation and the approximation using fast adaptive uniformization (see Section 4.3.3).

The first example that we consider is the transcription regulation of a repressor protein in bacteriophage λ, where we approximate the probability distribution at several time instances. In the second example, which is a gene expression network [108], we compute the distribution of the time until the number of produced proteins exceeds a certain threshold. In both examples the number of states reachable from the initial state is infinite. The number of chemical species is 6 and 2, and the number of chemical reactions is 10 and 4, respectively. We compare the running time of our numerical reachability analysis to that of the simulative approach for both examples, for different precision levels. Our results show that numerical approximation based on reachability analysis is superior to statistical estimation based on repeated simulation, especially if we increase the desired precision level. For instance, the numerical

approximation of the first example needs 39 minutes for a total approximation error of 2×10^{-5}, which distributes among all states. Simulation requires more than six hours if the statistical error of a single event is to be bounded by 10^{-5} and more than sixty hours for 10^{-6}.

Unbounded Range. For realistic systems, the state space of the Markov chain is extremely large, because its size grows exponentially in the number of involved chemical species. Moreover, if upper bounds on the state variables cannot derived from certain conservation laws, their range is assumed to be infinite although in practice the number of molecules is bounded. Then from the infinite structure, we can compute bounds that are kept with a very high probability. Even though every state in the infinite state space has a non-zero probability, certain attracting regions force most of the probability mass to remain within a finite range.

Example 5.1. *In Example 2.1, the degradation rates $\alpha_3(\mathbf{s})$ and $\alpha_4(\mathbf{s})$ grow linearly in the state variables. Thus, the higher the number of mRNA or protein molecules the more likely is their degradation. Depending on the rate constants $c_1, \ldots, c_4$, the system becomes "stable" in different regions. As time approaches infinity, the main part of the probability mass will be close to a region where production and degradation of molecules cancel each other out. Below, we discuss in general under which conditions the system approaches such a stable distribution.*

Holding Times and Jump Probabilities. A Markov chain $(\mathbf{X}(t))_{t\geq 0}$ defined in the way above is a *stable and conservative jump process* [9]. Thus, there exists a sequence of jump times $(\tau(k))_{k\in\mathbb{N}_0}$ and a sequence $\left(\hat{\mathbf{X}}(k)\right)_{k\in\mathbb{N}_0}$ of visited states such that

$$\tau(0) = 0 < \tau(1) < \tau(2) < \ldots \text{ and } \mathbf{X}(t) = \hat{\mathbf{X}}(k) \text{ if } \tau(k) \leq t < \tau(k+1).$$

The distribution of the k-th holding time $\tau(k+1) - \tau(k)$ under the condition $\hat{\mathbf{X}}(k) = \mathbf{s}$ is negative exponentially distributed with parameter $\lambda_{\mathbf{s}} = \sum_{j:\mathbf{s}\in G_j} \alpha_j(\mathbf{s})$, also called *exit rate* of state $\mathbf{s}$.

If the sum of all holding times is finite with positive probability, the Markov chain is said to *explode* and the limiting distribution does not exist. Explosive Markov chains are not of interest for the application area of this work since in this case the system "gets lost at infinity". It is possible to check if the Markov chain does not explode by using *Reuter's Criterion* [9]. For the remainder of our presentation we assume that the rate functions α_j are such that the Markov chain does not explode.

Assume that the k-th state of the Markov chain is $\mathbf{s}$, that is, $\hat{\mathbf{X}}(k) = \mathbf{s}$. If at least one transition class is enabled in $\mathbf{s}$, the successor state is $u_j(\mathbf{s})$ for some j with $\mathbf{s} \in G_j$. The probability of successor $u_j(\mathbf{s})$ is given by

$$Pr\Big(\hat{\mathbf{X}}(k+1) = u_j(\mathbf{s}) \,|\, \hat{\mathbf{X}}(k) = \mathbf{s}\Big) = \tfrac{\alpha_j(\mathbf{s})}{\lambda_{\mathbf{s}}}.$$

The holding times and the jump probabilities play an important role for the simulation of the Markov chain, which is used to estimate the probability of a certain events.

5.1.1 Statistical Estimation of Probabilities

In this section we shortly review the basic steps that have to be carried out to estimate the probability of a certain measurable event using stochastic simulation. Throughout this section, we will denote this event by A and its probability by γ. For the analysis of biological systems, the events of interest may be the marginal distributions or even the joint distributions of certain chemical species. For instance, A may have the form $X_i(t) = k$, that is, the number of type i molecules is k.

Estimates are obtained in two steps. In the first step, a certain number of simulation runs of the Markov chain have to be generated, and in the second step, the results of the simulation runs are analyzed.

Trajectory Generation. A realization of the Markov chain, also called *trajectory* or *run*, is the random sequence of states visited by the process. If trajectories are produced by a computer, *pseudo-random numbers* are used to artificially generate randomness [73]. The basic steps of producing a single trajectory that starts in the initial state $\mathbf{y}$ at time 0 are as follows:

1. Initialize time $t = 0$ and state $\mathbf{s} = \mathbf{y}$.
2. Generate the holding time h, i.e., a sample of a random variable being exponentially distributed with parameter $-\lambda_{\mathbf{s}}$.
3. Generate the successor state, i.e., a sample j of a discrete random variable Z that has probability distribution $P(Z = j) = \alpha_j(\mathbf{s})/\lambda_{\mathbf{s}}$.
4. Set $t = t + h$, $\mathbf{s} = u_j(\mathbf{s})$ and go to Step 2 if $t < T$.

In Step 2, we generate the holding time of the current state $\mathbf{s}$. Pseudo-random number generators usually draw from a uniform distribution. Thus, for a given random sample r_1 that is uniformly distributed on $(0, 1)$, we calculate an exponentially distributed sample by using the inverse transform method. More

precisely, we compute the inverse $-\frac{\ln r_1}{\lambda_{\mathbf{s}}}$ of the cumulative distribution function of the exponential distribution. In Step 3, the same idea is used to decide, which reaction occurs next. The inverse of the cumulative distribution function of Z is given by $j = \min\left\{j" : \sum_{j'=1}^{j"} \alpha_{j'}(\mathbf{s}) > r_2 \cdot \lambda_{\mathbf{s}}\right\}$, where r_2 is again a random sample that is uniformly distributed on $(0, 1)$. In the final step, the current time and the current state are updated. The simulation is terminated if the time horizon T of interest is reached and continued otherwise.

Output Analysis. The problem of estimating the probability γ of the event A can be reformulated as estimating the expectation of the random variable χ_A with

$$\chi_A(\omega) = \begin{cases} 1 & \text{if } \omega \in A, \\ 0 & \text{if } \omega \notin A, \end{cases}$$

where ω is a trajectory. The expectation $E[\chi_A]$ equals γ, since $E[\chi_A] = 1 \cdot Pr(\chi_A = 1) + 0 \cdot Pr(\chi_A = 0) = \gamma$. Therefore, we can resort to the standard estimation procedure for expectations. Assume that N is the number of runs that have been carried out and $Y_1, \ldots, Y_N$ are independent and identically distributed as χ_A. Thus, from the ρ-th run we get a realization of Y_ρ by checking if A has occurred or not. It is important to point out that we have to guarantee the independence of the Y_ρ's. This implies that we generate N independent trajectories of the Markov chain, each time with a different initial seed[1] for the pseudo-random number generator. The sample mean $\bar{Y} = \frac{1}{N}\sum_{\rho=1}^{N} Y_\rho$ is then an *unbiased* and *consistent estimator* [73] for $E[\chi_A]$. The former means that $E[\bar{Y}] = E[\chi_A]$ and the latter refers to the fact that as N increases the estimator $\bar{Y}$ becomes closer to γ. Note that $\bar{Y}$ is equal to the relative frequency of the event A. Let $\sigma^2 = VAR[\chi_A]$ be the variance of χ_A. We evaluate the quality of the estimator $\bar{Y}$ by applying the central limit theorem, which states that $\bar{Y}$ will approximately have a Normal distribution with mean $E[\chi_A] = \gamma$ and variance σ^2/N. Hence, for large N the random variable

$$Z = \frac{\bar{Y} - \gamma}{\sqrt{\sigma^2/N}}$$

has a standard Normal distribution, that is, the mean is zero and the variance is one. Knowing the distribution of Z enables us reason about the difference

[1] The seed of a pseudo-random number generator is an initial value, on which the sequence of generated numbers depend [73].

$|\bar{Y} - \gamma|$. Let $\beta \in [0,1]$ be the *confidence level* and $z \in \mathbb{R}^+$ such that $\beta = Pr(|Z| \leq z)$. Then

$$\beta = Pr(|Z| \leq z) = Pr\left(\frac{|\bar{Y} - \gamma|}{\sqrt{\sigma^2/N}} \leq z\right) = Pr\left(|\bar{Y} - \gamma| \leq z\sqrt{\sigma^2/N}\right).$$

We estimate σ^2 with the sample covariance $S^2 = \frac{1}{N-1}\sum_{\rho=1}^{N}(Y_\rho - \bar{Y})^2$, which is an unbiased estimator for σ^2. Then, for large N and a large number of realizations of the *confidence interval*

$$\left[\bar{Y} - z\sqrt{S^2/N}, \bar{Y} + z\sqrt{S^2/N}\right], \tag{5.1}$$

β is the fraction of intervals that cover γ. It therefore measures the quality of the estimator $\bar{Y}$.

For a practical application, two further remarks are important. Firstly, we usually choose $\beta \in \{0.95, 0.99\}$ and the corresponding value of z can be found in the table of the standard Normal distribution. Let Φ be the cumulative distribution function of the standard Normal distribution. Then, using that the Normal distribution is symmetric,

$$\Phi(z) = Pr(Z \leq z) = 1 - \tfrac{1-\beta}{2} = \tfrac{1+\beta}{2} \iff z = \Phi^{-1}\left(\tfrac{1+\beta}{2}\right).$$

Secondly, both, $\bar{Y}$ and S^2 can be computed efficiently if during the trajectory generation the realizations of the two sums $\sum_{\rho=1}^{N} Y_\rho$ and $\sum_{\rho=1}^{N} Y_\rho^2$ are calculated, since it can be easily shown that

$$S^2 = \frac{\sum_{\rho=1}^{N} Y_\rho^2}{N-1} - \frac{\left(\sum_{\rho=1}^{N} Y_\rho\right)^2}{(N-1)N}.$$

Thus, if $r \in \{0, \ldots, N\}$ is the number of times event A occurred during the N simulation runs, $\bar{Y} = r/N$ and $S^2 = \frac{r(N-r)}{N(N-1)}$.

If the interval in Equation (5.1) is large relative to $\bar{Y}$ the quality of the estimator is poor and more simulation runs have to be carried out. For our experimental results in Section 5.1.2, we fixed the relative width of the interval to be 0.2 (which means that we have a relative error of at most 0.1) and chose confidence level $\beta = 0.95$. Thus, $z \approx 1.96$ and we can determine the number of necessary runs by bounding the relative width

$$2 \cdot \frac{z \cdot \sqrt{S^2/N}}{\gamma} \leq 0.2 \implies \frac{z^2}{0.01}\frac{S^2}{\gamma^2} \leq N \implies 384 \cdot \frac{S^2}{\gamma^2} \leq N$$

Assume now that we want to estimate the probability of events that occur at least with probability γ. Using the fact that $\sigma^2 = VAR[\chi_A] = \gamma(1-\gamma)$ and

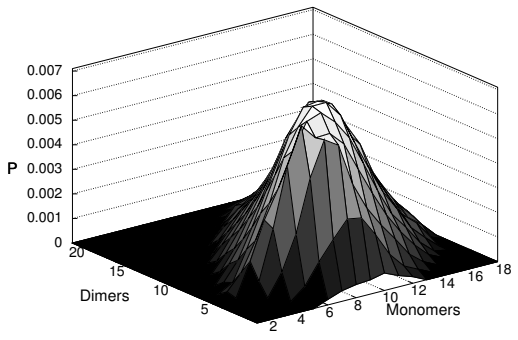


Figure 5.1: Probability distribution of monomers and dimers in the phage λ model.

replacing S^2 by σ^2 yields $N \geq 384 \cdot \frac{1-\gamma}{\gamma}$ [92]. Thus, estimating probabilities having at least the order of magnitude of 10^{-5}, for instance, with a relative error of 0.1 and a confidence of 95% requires at least $N = 38,000,000$ simulation runs.

5.1.2 Experimental Results

For our experimental results, we consider two examples from biology. One if a model for the transcription regulation of a repressor protein in bacteriophage λ [44]. This protein is responsible for maintaining lysogeny of the λ virus in E. coli [5]. We compute the full probability distribution for different precision levels. Our second example uses the gene expression model of Example 2.1. We calculate the distribution of the time until the number of produced proteins exceeds 500. The running times for simulations grow linearly with the number of simulations, and so for long number of simulations infer the running times by simple multiplication.

There is no one-to-one correspondence between the statistical accuracy of the estimates that we derive via simulation and the precision of the numerical method. However, by assuming that the smallest event probability that has to be estimated is γ all results of the simulation have a "precision" of at least γ. Intuitively, we simulate often enough to reason about events that occur with a probability of at least γ. We therefore refer to γ as the *single event error* (cf. Table 5.1 and 5.2). Note that the simulation results are still subject to the statistical errors since the true values may not be covered by the confidence interval (compare Section 5.1.1).

numerical approximation				Gillespie simulation		
running time	total approx. error	$\lvert S_l \rvert$	δ	running time	single event error	# runs
55 min 5 sec	3×10^{-6}	239792	10^{-15}	> 6000 h	10^{-8}	$> 3 \times 10^{10}$
39 min 16 sec	2×10^{-5}	187204	10^{-14}	> 500 h	10^{-7}	$> 3 \times 10^{9}$
25 min 2 sec	2×10^{-4}	140969	10^{-13}	67 h 22 min	10^{-6}	$> 3 \times 10^{8}$
15 min 41 sec	1×10^{-3}	101078	10^{-12}	6 h 44 min	10^{-5}	$> 3 \times 10^{7}$
6 min 33 sec	7×10^{-3}	67540	10^{-11}	40 min	10^{-4}	$> 3 \times 10^{6}$
3 min 12 sec	4×10^{-2}	40373	10^{-10}	4 min	10^{-3}	$> 3 \times 10^{5}$

Table 5.1: Comparison of the running times for the phage λ model.

The approximation error ϵ of the numerical method is the sum of the approximation error of *all* states in the Markov chain. Note that the probabilities of states not in S_l are underapproximated with zero and their true probabilities increase depending on how close they are to an attracting region. The error of a single state probability $p_{\mathbf{s}}^{(t)}$ is much smaller than ϵ but precise values for the single error are hard to obtain. A rough estimation of the single errors can be obtained by dividing the total error by the average size $|S_l|$ of the significant sets (cf. Table 5.1 and 5.2), even though ϵ may not be uniformly distributed on the significant set. On the other hand, ϵ also includes the error of insignificant states and, thus, distributes among much more states than only those in S_l.

Goutsias's model The Phage λ model involves 6 different species and 10 reactions, as shown in Example 4.2. The initial state of the system is given by $\mathbf{y} = (2, 6, 0, 2, 0, 0)$ and the time horizon is $t = 300$. We approximate the probability distributions of the underlying CTMC at 100 equidistant time instances. Figure 5.1 shows a plot of the distribution of dimers and monomers at time instant $t = 300$. In Table 5.1, we list the running times of our numerical method as well as the running time of the simulation. The column with header $|S_l|$ lists the average number of states in the sets $S_0, S_1, \ldots$ and δ is the threshold in Equation (3.1).

Gene Expression. For the transition classes of the gene expression example we refer to Example 2.1. For the rate constants, we choose $c_1 = 0.05$, $c_2 = 0.0058$, $c_3 = 0.0029$, and $c_4 = 10^{-4}$, where c_3 and c_4 correspond to a half-life of 4 minutes for mRNA and 2 hours for the protein [108]. We compute the probability that at least 500 proteins are in the system at 100 equidistant time instances. Figure 5.2 shows the cumulative probability distribution of the time until the number of proteins reaches 500 for the first time (note that eventually the threshold of 500 is reached with probability one). In Table 5.2, we list the results for the gene expression example, where, as above, $|S_l|$ denotes the average number of states in the sets $S_0, S_1, \ldots$ and δ is the threshold in Equation (3.1).

Discussion. Even if we consider the total approximation error ϵ as a rough bound for the single error of each state probability, thus favoring simulation, the speed-up factor of the numerical approximation is large, especially if the precision increases. The necessary precision level up to which probability distributions are approximated may depend on the system under study. It is, however, important to note that the occurrence of rare biochemical events can have important effects. For instance, the spontaneous, epigenetic switching rate from

numerical approximation				Gillespie simulation		
running time	total approx. error	$\lvert S_l \rvert$	δ	running time	single event error	# runs
4.2 sec	5×10^{-6}	9816	10^{-12}	> 500 h	10^{-7}	$> 3 \times 10^9$
3.6 sec	5×10^{-5}	8719	10^{-11}	> 50 h	10^{-6}	$> 3 \times 10^8$
3.0 sec	5×10^{-4}	7516	10^{-10}	5 h 3 min	10^{-5}	$> 3 \times 10^7$
2.4 sec	4×10^{-3}	6265	10^{-9}	30 min 18 sec	10^{-4}	$> 3 \times 10^6$
1.9 sec	4×10^{-2}	4939	10^{-8}	3 min sec	10^{-3}	$> 3 \times 10^5$

Table 5.2: Comparison of the running times for the gene expression example.

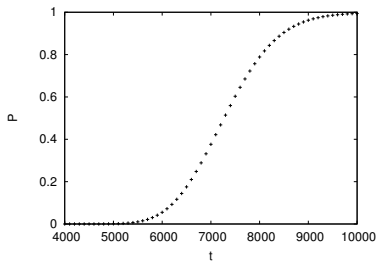


Figure 5.2: Cumulative probability distribution of the time until the number of proteins reaches 500 for the first time in the gene expression example.

the lysogenic state to the lytic state in phage λ-infected E. coli is experimentally estimated to be in the order of 10^{-7} per cell per generation [74].

5.1.3 Conclusion

We have demonstrated that, for the computation of event probabilities, a numerical reachability analysis provides an efficient alternative to simulation-based methods.

Even though simulation is widely used, the advantages of numerical methods increase as more sophisticated techniques become available. They reduce the computational effort, especially if accurate results are desired. Moreover, for the calibration of parameters many instances of the model have to be solved and in this case short running times for a single solution are necessary.

Until now we have analyzed examples of intrinsically stochastic systems that have been published in the literature. As future work, we are planning to apply our numerical reachability algorithm in collaboration with experimentalists working on new stochastic models. Moreover, we are planning to combine our numerical method with parameter estimation techniques.

Standard numerical reachability analysis methods are inefficient for large state spaces (in the case of high dimension and/or many molecules) and inapplicable for unbounded state spaces, and thus one resorts to simulation. We have demonstrated that certain optimization techniques from computer science - localization, on the fly abstraction - put many examples within the reach of numerical reachability analysis. Indeed, when high accuracy is required these methods outperform simulation-based techniques.

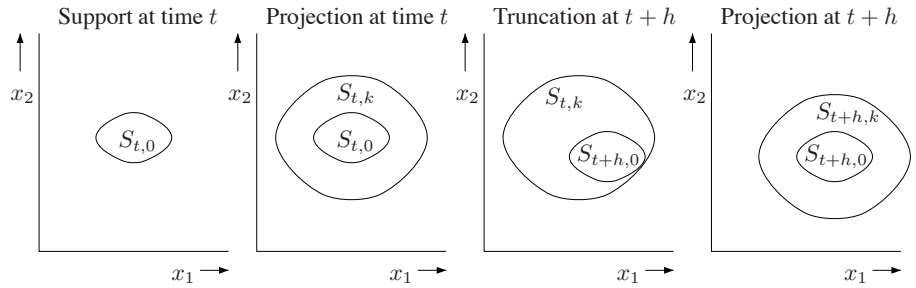


Figure 5.3: Projection construction and truncation in the original FSP approach.

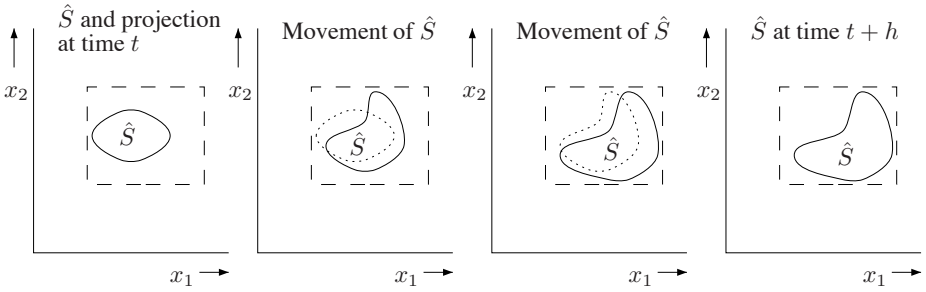


Figure 5.4: Projection (dashed line) in the improved FSP vs. significant set $\mathcal{S}_k$ in FAU.

5.2 Finite Space Projection

The finite state projection (FSP) algorithm is an alternative approach for the solution of the chemical master equation on a reduced state space. The basic idea in the original version of the algorithm is to bound the number of reactions that occur in an interval $[t, t+h]$ by k, where k is chosen w.r.t. a given precision ϵ (see [82, Chapter 5.3]). The idea is illustrated in Figure 5.3 for a two-dimensional phase space. Assume that at time t only states in the set $S_{t,0}$ have positive probability, i.e. $S_{t,0}$ is the support of the distribution. The master equation is solved for all states in the set $S_{t,k}$, which is the set of all states reachable from $S_{t,0}$ within at most k transitions. After the computation the probabilities of all states in $S_{t,k}$ are known and it is possible to truncate $S_{t,k}$ yielding the set $S_{t+h,0}$. This truncation is done in such a way that the remaining probability mass is greater than a given threshold. In order to proceed further in time, the same procedure can be repeated until the final time instant is reached. As opposed to that, the FAU approach is based on a dynamic set $\mathcal{S}_k$ that may change in each of the R iteration steps that are performed to solve the master

equation for the interval $[t, t+h]$. Obviously, k and R are of the same order of magnitude for a given upper bound on the probability mass that is lost, but the total number of times probability mass is moved from one state to another may be significantly smaller for the FAU approach. The reason for this is that FAU considers fewer states in total and many states are only considered during some of the R iteration steps. In particular, if the probability mass is moving in a certain direction (as it is typically the case during the transient phase), then certain states are "left behind". The original FSP approach, on the other hand, constructs a projection independent of the direction in which the probability mass is moving. For small examples or those that are almost in steady-state, the benefit of the FAU method may be neglected, but in all other cases the FAU method requires less computational work and is therefore faster. The accuracy of the two methods is similar because during the FAU method only states with very low probability are neglected. The FSP method, however, has the advantage that a-priori specified error bounds hold whereas the error of the FAU method depends on the chosen threshold δ and is determined after the computation. For our experimental results, we controlled the total error by choosing values for δ that range from 10^{-15} to 10^{-11}. For chemical reaction networks, this leads to sufficiently accurate results except one is interested in rare events that occur with a probability smaller than δ. It is important to note that if the Markov process is simulated using, for instance, Gillespie's algorithm, on average $O(1/\delta)$ simulation runs are necessary to observe such events [22].

Recently, improvements to the FSP method have been suggested where smaller projections are constructed by heuristically determining the direction in which the probability mass is moving [82, 84]. The idea is to use a rectangular projection and to introduce "sink states" on all boundaries. If the sink state on a certain boundary of the projection receives a significant amount of probability mass, the projection for the next time step is extended in the corresponding direction. The projection is not extended in those directions where the sink states have small probability. This improvement leads to smaller memory requirements for the FSP method. On the other hand, for this variant of the algorithm the error cannot be specified apriori since the probability mass that gets lost can only be determined after the computation is done. Of course, it is possible to repeat the computation with a larger projection until an apriori specified error bound is met. Still the memory requirements are larger than for FAU because the projection is fixed during each time step whereas in the FAU method the set of considered states is updated in an on-the-fly fashion. We illustrate the difference between the improved FSP method and the FAU method

in Figure 5.4 for a two-dimensional system. Here, the dashed line represents the projection in the FSP approach for the interval $[t, t+h]$ and the solid line represents the set $\mathcal{S}_k$ of significant states in the FAU approach. During $[t, t+h]$, the (possibly non-convex) set $\mathcal{S}_k$ changes in each of the R iterations. We use dotted lines for $\mathcal{S}_k$ in the previous iteration step and solid lines for the current step.

We have implemented both, the original FSP approach and its improved variant. For our examples, the memory requirements of the original approach become intractable after few time steps. For the improved FSP approach the crystallization and the signalling case studies could not finish the computation due to time and, respectively, space constraints. In the case of crystallization, the system times out after 10 hours. The reason is that in each time step the projection is extended by a fixed amount, and it is unable to follow the dynamics of the system. More precisely, the dynamics are very fast at the beginning (large projections are needed to ensure that enough probability mass remains within the projection during a time step of length h) and become slow after many dimerizations. Thus, at later time steps small projections are sufficient. We tried different choices for the time step h and found that $h = 0.1$ performs best. For this choice after 10 hours, 15% of the time horizon was reached. For the signalling example, the FSP algorithm runs out of memory even though the projection was optimized as a rectangular window with the bounds

$$[0,30] \times [870,900] \times [0,30] \times [0,17] \times [0,11] \times [0,4] \times [0,2] \times [0,1] \times [0,1] \times [9996,10000] \times [0,1] \times [0,4].$$

This rectangular box in 12 dimensions contains about 4 million states. FAU does not use a rectangular window but if we compute the maximal and minimal molecule numbers in the set $\mathcal{S}_k$ for each dimension we get

$$[0,15] \times [870,885] \times [5,30] \times [0,21] \times [0,11] \times [0,6] \times [0,4] \times [0,2] \times [0,1] \times [9983,10000] \times [0,1] \times [0,17].$$

The number of states in $\mathcal{S}_k$, however, is only 58311. The reason is that many states within these bounds are dropped because their probability is smaller than δ. Finally, we could run FSP for the phage λ model. For an error of 7e−3, the running time of FSP is 69 minutes, while for an error of 3e−6, the algorithm takes 7.5 hours (compare with the first and last lines of Table 4.5). It is important to note that the number of states in the projection that have a probability smaller than, say, 10^{-15} is significant and it is difficult to develop heuristics that optimize the shape of the projection in such a way that systems

with higher dimensions can be solved efficiently with the FSP approach. The main advantage of FAU is that it does not use minimal and maximal bounds on the molecule numbers but decides for each individual state whether it should be considered or not.

5.3 Krylov Subspace Method

Krylov subspace methods are widely used for large eigenvalue problems, for solving linear equation systems, and also for approximating the product of a matrix exponential and a vector [36, 97]. We are interested in the latter approximation and show how it can be used to solve the CME, either in a global fashion or in combination with the sliding window method. Recently, Krylov subspace methods have been applied to the CME by Sidje et al. [12].

5.3.1 Global Krylov Subspace Method

Recall that a global solution of the CME is given by $\mathbf{p}^{(t)} = \mathbf{p}^{(0)} e^{Qt}$. In the sequel, we describe the approximation of $e^{tA}\boldsymbol{v}$, where A is a $N \times N$ square matrix and $\boldsymbol{v}$ is a column vector of length N. We obtain an approximation of $\mathbf{p}^{(t)}$ by choosing $A = (Q)^T$ and $\boldsymbol{v} = (\mathbf{p}^{(0)})^T$.

Let us first assume that $t = 1$. The main idea is to generate a basis of the m-th Krylov subspace

$$K_m = Span\left\{\boldsymbol{v}, A\boldsymbol{v}, \ldots, A^{m-1}\boldsymbol{v}\right\},$$

and to seek an approximate solution for $e^A\boldsymbol{v}$ from this subspace. Let $q^{\min}$ be the nonzero monic polynomial of lowest degree such that $q^{\min}(A)\boldsymbol{v} = 0$. We choose $m \in \mathbb{N}$ such that the degree of $q^{\min}$ is greater or equal to m. In this case, the vectors $\boldsymbol{v}, A\boldsymbol{v}, \ldots, A^{m-1}\boldsymbol{v}$ are linearly independent, and for every element $\boldsymbol{x} \in K_m$ there exists a polynomial q of degree at most $m-1$ with $\boldsymbol{x} = q(A)\boldsymbol{v}$. Note that in practice we choose $m = 30$ or $m = 20$, because the degree of $q^{\min}$ is usually greater than 30. However, if not, the problem can be solved *exactly* in the d-th Krylov subspace, where d is the degree of $q^{\min}$. Working directly with the basis $\left\{\boldsymbol{v}, A\boldsymbol{v}, \ldots, A^{m-1}\boldsymbol{v}\right\}$ is numerically unstable. Therefore, we construct an orthonormal basis $\{\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_m\}$ for K_m by applying Arnoldi's algorithm to $\boldsymbol{v}, A\boldsymbol{v}, \ldots, A^{m-1}\boldsymbol{v}$. Let H_m be the $m \times m$ upper Hessenberg matrix computed

by the Arnoldi algorithm and let $h_{m+1,m}$ be the last normalization value. By $\mathcal{M}_m$ we denote the $N \times m$ matrix with column vectors $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_m$. Then

$$\begin{aligned} (a) \qquad A\mathcal{M}_m &= \mathcal{M}_m H_m + h_{m+1,m}\boldsymbol{v}_{m+1}\boldsymbol{e}_m^T, \\ (b) \quad \mathcal{M}_m^T A\mathcal{M}_m &= H_m, \end{aligned} \tag{5.2}$$

where $\boldsymbol{e}_k$ is a column vector of appropriate size whose k-th component is one and all other components are zero. Intuitively, Equation (5.2)(b) states that H_m is the matrix projection of A onto K_m w.r.t. the basis defined by $\mathcal{M}_m$. An approximation of $e^A\boldsymbol{v}$ in K_m expressed using $\mathcal{M}_m$ is $e^A\boldsymbol{v} \approx \mathcal{M}_m\mathbf{y}$, where $\mathbf{y}$ is a vector of size m. We choose

$$\mathbf{y} = ||\boldsymbol{v}||_2 \, e^{H_m}\boldsymbol{e}_1, \tag{5.3}$$

which yields the approximation error [36]

$$\left|\left|e^A\boldsymbol{v} - ||\boldsymbol{v}||_2 \, \mathcal{M}_m e^{H_m}\boldsymbol{e}_1\right|\right|_2 \leq 2\,||\boldsymbol{v}||_2 \, \frac{\rho^m e^\rho}{m!}, \tag{5.4}$$

where $\rho = ||A||_2$ is the spectral norm of A. The approximation in Equation (5.3) still involves the computation of the matrix exponential of H_m, but as H_m is of small dimension and has a particular structure (upper Hessenberg), this requires a smaller computational effort. For the matrix exponential of small matrices, methods such as Schur decomposition and Padé approximants may be applied [81].

Assume now that the time instant t is arbitrary, i.e., we want to approximate $e^{tA}\boldsymbol{v}$ for some $t > 0$. In order to control the approximation error, we calculate $e^{tA}\boldsymbol{v}$ stepwise by exploiting that $e^{(h_1+h_2)A}\boldsymbol{v} = e^{h_1 A} \cdot e^{h_2 A}\boldsymbol{v}$ for $h_1, h_2 \geq 0$. For a step size h, we approximate $e^{hA}\boldsymbol{v}$ by $||\boldsymbol{v}||_2 \, \mathcal{M}_m e^{hH_m}\boldsymbol{e}_1$ because the Krylov subspaces associated with A and hA are identical and $\mathcal{M}_m^T(hA)\mathcal{M}_m = hH_m$. It follows from Equation (5.4) that we have a small error bound if $||Ah||_2$ is small.

To summarize, the Krylov subspace method approximates $e^{At}\boldsymbol{v}$ by an iteration stepping forward in time with dynamically chosen step sizes $h_1, h_2, \ldots$. In each iteration step, we compute a vector

$$\boldsymbol{u}_i = ||\boldsymbol{u}_{i-1}||_2 \, \mathcal{M}_m^{(i)} e^{h_i H_m^{(i)}}\boldsymbol{e}_1,$$

where initially $\boldsymbol{u}_0 = \boldsymbol{v}$. The matrices $\mathcal{M}_m^{(i)}$ and $H_m^{(i)}$ result from the i-th execution of Arnoldi's algorithm for the construction of an orthonormal basis of the

subspace $Span\left\{\boldsymbol{u}_{i-1}, A\boldsymbol{u}_{i-1}, \ldots, A^{m-1}\boldsymbol{u}_{i-1}\right\}$. When the elapsed time equals t, we obtain an approximation of $e^{At}\boldsymbol{v}$.

For the step size of the Krylov subspace method, a popular heuristic is to choose h_{i+1} depending on an estimate of the error ϵ_i of the previous step. Let $tol > 0$ be an a priori specified tolerance. A common scheme consists of three steps [89]. (1) Define $h_i = 0.9(\frac{tol}{\epsilon_{i-1}})^{1/m}h_{i-1}$, (2) compute $\boldsymbol{u}_i$ and the error estimation ϵ_i. (3) If $\epsilon_i > 1.2\ tol$ reject $\boldsymbol{u}_i$, replace ϵ_{i-1} with ϵ_i, and go to step (1). For the experimental results in Section 5.1.2, we used the Expokit software [102] where the small exponential, e^{hH_m}, is computed via the irreducible Padé approximation [6].

5.3.2 Local Krylov Subspace Method

Assume now that we use the Krylov subspace method in line 10 of the sliding window algorithm (Algorithm 4.1), to approximate $\hat{\boldsymbol{p}}^{(t_{r-1})} \cdot e^{Q_r \cdot h_r}$ (cf. Equation (2.8)). By letting $\boldsymbol{v} = (\hat{\boldsymbol{p}}^{(t_{r-1})})^T$, $A = Q_r^T$, and $t = h_r$ we can apply the same procedure as in the global case. Note that this yields a nested iteration because the time steps h_r are usually much bigger than the time steps of the Krylov subspace method. For the Krylov subspace method, using the matrix Q_r instead of Q offers important advantages. The Arnoldi process is faster as Q_r usually contains fewer nonzero entries than Q. As well, the sliding window method is likely to provide matrices with a smaller spectral norm $||Q_r||_2$. This allows for bigger time steps during the Krylov approximation, as can be seen in our experimental results in 4.2.6.

Part II

Hybrid Propagation Models for Biochemical Reaction Networks

Introduction

In Section 2.3, we have seen how to construct a propagation model for the chemical master equation of a biochemical reaction network. This computational model is propagating probabilities through the state space of a Markov chain. However, the framework of propagation models is not restricted to this kind of differential equations and in this part we will construct propagation processes for stochastic hybrid models where both probabilities and variable expectations are propagated through the state space.

If the system under consideration contains large populations, then the numerical algorithms introduced in the first part of this thesis perform poorly. The reason is that the random variables that represent large populations have a large variance, even though their *relative* variance (i.e., relative to their expectation) is often small. Thus, a large number of states have a significant probability, which renders the numerical approximation of the distribution computationally expensive or infeasible. Motivated by this observation, in this part of the thesis, instead of approximating the solution of a CME, we approximate an aggregated solution of the CME. We propose three such approximation methods, for each of which we give different definitions of the aggregated solution. However, all of these definitions have in common the fact that they contain two components: one that represents a probability distribution over an aggregated state space, and one that represents variable expectations of abstract states of an aggregated state space.

Each of the three approaches is based on the construction of a stochastic hybrid model in which certain discrete random variables of the original propagation model are approximated by continuous deterministic variables. We compute the solution of the stochastic hybrid models using the RK4 numerical algorithm that discretizes time and in each step performs a mutual update of the transient probability distribution of the discrete stochastic variables and the values of the continuous deterministic variables.

For the first two algorithm we present experiments done on biological case studies, while for the last one, we only give a sketch of the algorithm and we present an intermediate result which we believe to be essential in the design of the complete algorithm.

Chapter 6

Preliminaries

6.1 Aggregation

6.1.1 Aggregated state space.

Recall that for a transition class model M, the stochastic vector $\boldsymbol{p}^{(t)}$ is the transient solution of the associated continuous-time Markov chain $(\mathbf{X}(t))_{t\geq 0}$ at time t, and that this vector has entries $p_{\mathbf{s}}^{(t)}$, that correspond to the probability of being in state $\mathbf{s}$ at time t. However, sometimes it is not the probabilities $p_{\mathbf{s}}^{(t)}$ that are of interest, but the probabilities to be in a certain subset of states.

For a subset $A \subseteq \mathcal{S}$, we define $p_A^{(t)} = \sum_{\mathbf{s}\in A} p_{\mathbf{s}}^{(t)}$. Furthermore, given the Markov chain $\mathbf{X}(t)$, and a partition[1] $\hat{\mathcal{S}}$ of the state space $\mathcal{S}$ we define the stochastic process $\left(\hat{\mathbf{X}}(t)\right)_{t\geq 0}$ over the state space $\hat{\mathcal{S}}$ as follows: $\hat{\mathbf{X}}(t) = A$ if and only if $\mathbf{X}(t) = \mathbf{s}$ and $\mathbf{s} \in A$. We note that $p_A^{(t)} = Pr\left(\hat{\mathbf{X}}(t) = A\right) =: \hat{p}_A^{(t)}$.

We refer to the states $A \in \hat{\mathcal{S}}$ of the process $\hat{\mathbf{X}}(t)$ as *abstract states*, as opposed to the states $\mathbf{s} \in \mathcal{S}$ of the process $\mathbf{X}(t)$ that are *concrete states*.

6.1.2 Aggregated PM

For theoretical purposes only, we construct a propagation model $\hat{N}$ whose continuous-time propagation process is the exact equivalent of the transient probability vectors $\hat{\mathbf{p}}^{(t)}$, of the aggregated stochastic process $\hat{\mathbf{X}}(t)$.

[1] A partition $\hat{\mathcal{S}}$ of $\mathcal{S}$ is such that $\mathcal{S} = \bigcup_{A\in\hat{\mathcal{S}}} A$ and $A \cap A' = \emptyset$ for all $A, A' \in \hat{\mathcal{S}}$, and $A \neq A'$. From now on, $\hat{\mathcal{S}}$ will always denote a partition of the state space $\mathcal{S}$.

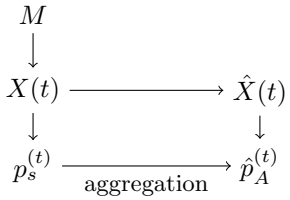


Figure 6.1: Aggregation scheme starting with the $TCM\ M$. An TCM equivalent to $\hat{\mathbf{X}}(t)$ can not be constructed.

For the $TCM\ M = \langle \mathcal{S}, \mathbf{y}, \{\mathcal{C}_1, \ldots \mathcal{C}_m\}\rangle$ and a partition $\hat{\mathcal{S}}$ of $\mathcal{S}$, we construct a PM $\hat{N}$, with state space $\hat{\mathcal{S}}$ and mass space $\mathcal{M}$, the set of sub-stochastic vectors over the state space $\mathcal{S}$. The continuous-time propagation process $\hat{\mathbf{g}}^{(t)}$ of $\hat{N}$ behaves such as:

$$\hat{p}_A^{(t)} = \sum_{\mathbf{s}\in A} \hat{g}_{A,\mathbf{s}}^{(t)}, \forall t \geq 0, A \in \hat{\mathcal{S}}.$$

Note that, the valuation of the propagation process $\hat{\mathbf{g}}^{(t)}$ on state $A \in \hat{\mathcal{S}}$ is an element of the mass space $\mathcal{M}$, that is, a sub-stochastic vectors over $\mathcal{S}$. And thus, the notation $\hat{g}_{A,\mathbf{s}}^{(t)}$ refers to the value in state $\mathbf{s} \in \mathcal{S}$ of the sub-stochastic vector $\hat{g}_A^{(t)}$.

Definition 6.0.1 (Aggregated PM)**.** *The* aggregated propagation model $\hat{N}$ *of a* $TCM\ M = \langle \mathcal{S}, \mathbf{y}, \{\mathcal{C}_1, \ldots \mathcal{C}_m\}\rangle$*, with respect to the partition* $\hat{\mathcal{S}}$ *is defined as the tuple* $\left\langle \hat{\mathcal{S}}, \mathcal{M}, \zeta, \pi \right\rangle$*:*

- $\mathcal{M} \subseteq [\mathcal{S} \to [0,1]]$*, and for all* $\mu \in \mathcal{M}$ *we have that* $\sum_{\mathbf{s}\in\mathcal{S}} \mu(s) \leq 1$*,*
- *the initial mass vector* ζ *is:*

$$\zeta_A = \begin{cases} \boldsymbol{e}_{\mathbf{y}}, & \text{if } \mathbf{y} \in A, \\ 0, & \text{otherwise,} \end{cases}$$

  where $\boldsymbol{e}_{\mathbf{s}}$ *is the unity vector, for which:* $\boldsymbol{e}_{\mathbf{s}}(\mathbf{s}') = 1$ *only for* $\mathbf{s}' = \mathbf{s}$*, and is* 0 *otherwise.*

- *the edge function* π*, for edge* $A \to A'$ *and sub-stochastic vector* μ *is:*

$$\pi_{A\to A'}(\mu) = \sum_{\mathbf{s}\in A, u_j(\mathbf{s})\in A'} \mu(s) \cdot \alpha_j(\mathbf{s}) \cdot \boldsymbol{e}_{u_j(\mathbf{s})}.$$

We now prove the relation between the propagation model $\hat{N}$ and the aggregated stochastic process $\hat{X}(t)$.

Lemma 6.1. *If $\hat{N}$ is the derived PM of M with respect to partition $\hat{\mathcal{S}}$, then:*

$$\hat{p}_A^{(t)} = \sum_{\mathbf{s}\in A} \hat{g}_{A,\mathbf{s}}^{(t)}, \forall t \geq 0, A \in \hat{\mathcal{S}}.$$

Proof. In order to prove this, we will show that

$$\frac{dp_{\mathbf{s}}^{(t)}}{dt} = \frac{d\hat{g}_{A,\mathbf{s}}^{(t)}}{dt}, \forall t \geq 0, \mathbf{s} \in A, A \in \hat{\mathcal{S}},$$

from which our lemma follows directly.

Note that $\hat{g}_A^{(t)}$ is a sub-stochastic distribution over the states in A, and that $\hat{g}_{A,\mathbf{s}}^{(t)}$ is the probability to be in state $\mathbf{s}$ at time t.

First we have that:

$$\begin{aligned}
&\sum_{A\in\hat{\mathcal{S}}} \pi_{A\to A'} \hat{g}_{A,\mathbf{s}'}^{(t)} = \\
&= \sum_{A\in\hat{\mathcal{S}}} \left(\sum_{\substack{\mathbf{s}\in A \\ u_j(\mathbf{s})\in A'}} p_{\mathbf{s}}^{(t)} \cdot \alpha_j(\mathbf{s}) \cdot \boldsymbol{e}_{u_j(\mathbf{s})} \right)_{\mathbf{s}'} = \\
&= \sum_{\mathbf{s}\in S, u_j(\mathbf{s})=\mathbf{s}'} p_{\mathbf{s}}^{(t)} \cdot \alpha_j(\mathbf{s}).
\end{aligned}$$

And so:

$$\begin{aligned}
\frac{d\hat{g}_{A,\mathbf{s}'}^{(t)}}{dt} &= \sum_{A\in\hat{\mathcal{S}}} \pi_{A\to A'} \hat{g}_{A,\mathbf{s}'}^{(t)} - \sum_{A\in\hat{\mathcal{S}}} \pi_{A'\to A} \hat{g}_{A',\mathbf{s}'}^{(t)} + \pi_{A'\to A'} \hat{g}_{A',\mathbf{s}'}^{(t)} \\
&= \frac{dp_{\mathbf{s}'}^{(t)}}{dt}.
\end{aligned}$$

□

The propagation model $\hat{N}$ has the same amount of information about the original PM M as N. So, so far, no information has been lost during this aggregation scheme. In the next Section we present the theory of reaction rate equations, which aggregates the entire system into a single state.

Later on, we introduce different propagation models $\tilde{N}$ that choose different compromises between the amount of information to keep about the state of the propagation process, the accuracy of their approximation, and the speed of their

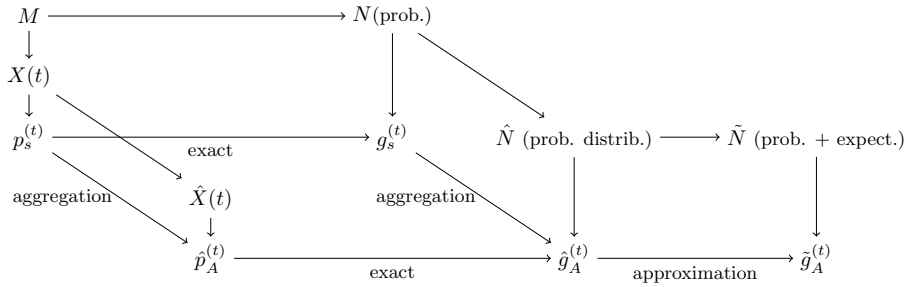


Figure 6.2: Relation between different propagation models (N, $\hat{N}$, $\tilde{N}$) and the original TCM M. N is the propagation model that corresponds to M, $\hat{N}$ corresponds to the aggregated stochastic process $\hat{X}(t)$, and $\tilde{N}$ refers to one of the three approximations that we propose in this second part of the thesis. Each PM has, under parenthesis, the type of mass that it propagates.

algorithm. The purpose of constructing the model $\hat{N}$ is to theoretically see what kind of object, over state space $\hat{\mathcal{S}}$ is approximated by our heuristic.

6.2 Reaction Rate Equation

A straightforward consequence of the CME is that the time derivative of the populations' expectations are given by

$$\frac{d}{dt}E[\mathbf{X}(t)] = \sum_{j=1}^{m} \mathbf{d}_j \cdot E\left[\alpha_j\left(\mathbf{X}(t)\right)\right]. \tag{6.1}$$

If all reactions of the system involve at most one reactant, Equation (6.1) can be simplified to

$$\frac{d}{dt}E[\mathbf{X}(t)] = \sum_{j=1}^{m} \mathbf{d}_j \cdot \alpha_j\left(E[\mathbf{X}(t)]\right).$$

because the propensity functions α_j are linear in $\mathbf{s}$. But in the case of bimolecular reactions, we have either $\alpha_j(\mathbf{s}) = c_j \cdot s_i \cdot s_\ell$ for some i, ℓ with $i \neq \ell$ or $\alpha_j(\mathbf{s}) = c_j \cdot s_i \cdot (s_i - 1)/2$ if the j-th reaction involves two reactants of type i. But this means that

$$E\left[\alpha_j\left(\mathbf{X}(t)\right)\right] = c_j \cdot E\left[X_i(t) \cdot X_\ell(t)\right]$$

or

$$E\left[\alpha_j\left(\mathbf{X}(t)\right)\right] = \frac{1}{2} \cdot c_j \cdot E\left[\left(X_i(t)\right)^2\right] - E\left[\left(X_i(t)\right)\right],$$

respectively. In both cases new differential equations are necessary to describe the unknown values of $E[X_i(t) \cdot X_\ell(t)]$ and $E\left[(X_i(t))^2\right]$. This problem repeats and leads to an infinite system of ODEs.

However, we can exploit Equation (6.1) to derive a deterministic limit of the stochastic model. Here, we shortly recall the basic steps of the derivation and for a detailed discussion, we refer to Kurtz [70].

6.2.1 Derivation of the Deterministic Limit

As rigorously derived by Gillespie [39], each reaction type has an associated *propensity function*, denoted by $\alpha_1, \ldots, \alpha_m$, which is such that $\alpha_j(\mathbf{s}) \cdot dt$ is the probability that, given $\mathbf{X}(t) = \mathbf{s}$, one instance of the j-th reaction occurs within $[t, t + dt)$. Recall that, for $j \in \{1, \ldots, m\}$ we have that $\mathbf{d}_j \in \mathbb{Z}^n$ is the *change vector* of the j-th reaction type, that is, $u_j(\mathbf{s}) = \mathbf{s} + \mathbf{d}_j$. Let $\mathbf{d}_j = \mathbf{d}_j^- + \mathbf{d}_j^+$ where $\mathbf{d}_j^-$ contains only non-positive entries, which specify how many molecules of each species are consumed (*reactants*) if an instance of the reaction occurs. The value $\alpha_j(\mathbf{s})$ is proportional to the number of distinct reactant combinations in state $\mathbf{s}$. More precisely, if $\mathbf{s} = (s_1, \ldots, s_n)$ is a state for which $\mathbf{s} + \mathbf{d}_j^-$ is nonnegative then

$$\alpha_j(\mathbf{s}) = \begin{cases} c_j & \text{if } \mathbf{d}_j^- = (0, \ldots, 0), \\ c_j \cdot s_i & \text{if } \mathbf{d}_j^- = -\mathbf{e}_i, \\ c_j \cdot s_i \cdot s_\ell & \text{if } \mathbf{d}_j^- = -\mathbf{e}_i - \mathbf{e}_\ell, \\ c_j \cdot \binom{s_i}{2} = c_j \cdot \frac{s_i \cdot (s_i - 1)}{2} & \text{if } \mathbf{d}_j^- = -2 \cdot \mathbf{e}_i, \end{cases} \tag{6.2}$$

where $i \neq \ell$, $c_j > 0$ is a constant, and $\mathbf{e}_i$ is the vector with the i-th entry 1 and all other entries 0. We set $\alpha_j(\mathbf{s}) = 0$ whenever the vector $\mathbf{s} + \mathbf{d}_j^-$ contains negative entries, that is, when not enough reactant molecules are available. The constant c_j refers to the rate at which a randomly selected pair of reactants collides and undergoes the j-th chemical reaction. Thus, if $\mathcal{N}$ is the volume (in liters) times Avogadro's number, then c_j

- scales inversely with $\mathcal{N}$ in the case of two reactants,
- is independent of $\mathcal{N}$ in the case of a single reactant,
- is proportional to $\mathcal{N}$ in the case of no reactants.

Since reactions of higher order (requiring more than two reactants) are usually the result of several successive lower order reactions, we do not consider the case of more than two reactants.

We first define a set of functions β_j such that if $\mathcal{N}$ is large then the propensity functions can be approximated as $\alpha_j(\mathbf{s}) \approx \mathcal{N} \cdot \beta_j(\mathbf{z})$, where $\mathbf{z} = (z_1, \ldots, z_n) = \mathbf{s} \cdot \mathcal{N}^{-1}$ corresponds to the vector of concentrations of chemical species and belongs to $\mathbb{R}^n$. Recall the dependencies of c_j on the scaling factor $\mathcal{N}$ as described above. For constants $k_j > 0$ that are *independent* of $\mathcal{N}$,

– $c_j = k_j \cdot \mathcal{N}$ in the case of no reactants,
– $c_j = k_j$ in the case of a single reactant,
– $c_j = k_j/\mathcal{N}$ in the case of two reactants.

From this, it follows that except for the case of bimolecular reactions, we can construct the functions β_j such that $\alpha_j(\mathbf{s}) = \mathcal{N} \cdot \beta_j(\mathbf{z})$.

$$\beta_j(\mathbf{z}) = \frac{\alpha_j(\mathbf{s})}{\mathcal{N}} = \begin{cases} \frac{c_j}{\mathcal{N}} = k_j & \text{if } \mathbf{d}_j^- = (0, \ldots, 0), \\ c_j \cdot \frac{s_i}{\mathcal{N}} = k_j \cdot z_i & \text{if } \mathbf{d}_j^- = -\mathbf{e}_i, \\ c_j \cdot s_i \cdot \frac{s_\ell}{\mathcal{N}} = k_j \cdot z_i \cdot z_\ell & \text{if } \mathbf{d}_j^- = -\mathbf{e}_i - \mathbf{e}_\ell, \end{cases}$$

where $i \neq \ell$. In the case of bimolecular reactions ($\mathbf{d}_j^- = -2 \cdot \mathbf{e}_i$), we use the approximation

$$\begin{aligned} \mathcal{N} \cdot \beta_j(\mathbf{z}) &= k_j \cdot \mathcal{N} \cdot z_i^2 = k_j \cdot s_i \cdot z_i = (\frac{1}{2} c_j \mathcal{N}) \cdot s_i \cdot z_i \\ &= \frac{1}{2} c_j \cdot s_i^2 \approx \frac{1}{2} c_j \cdot s_i(s_i - 1) = \alpha_j(\mathbf{s}), \end{aligned} \tag{6.3}$$

which is accurate if s_i is large, In order to derive the deterministic limit for the vector $\mathbf{X}(t) = (X_1(t), \ldots, X_n(t))$ that describes the chemical populations, we first write $\mathbf{X}(t)$ as

$$\mathbf{X}(t) = \mathbf{X}(0) + \sum_{j=1}^{m} \mathbf{d}_j \cdot C_j(t),$$

where $\mathbf{X}(0)$ is the initial population vector and $C_j(t)$ denotes the number of occurrences of the j-th reaction until time t. The process $C_j(t)$ is a counting process with intensity $\alpha_j(\mathbf{X}(t))$ and it can be regarded as a Poisson process whose time-dependent intensity changes according to the stochastic process $\mathbf{X}(t)$. Now, recall that a Poisson process $\tilde{Y}(t)$ with time-dependent intensity $\lambda(t)$ can be transformed into a Poisson process $Y(u)$ with constant intensity one, using the simple time transform $u = \int_0^t \lambda(h)dh$, that is, $Y(u) = Y(\int_0^t \lambda(h)dh) = \tilde{Y}(t)$. Similarly, we can describe $C_j(t)$ as a Poisson process with intensity one, i.e.,

$$C_j(t) = Y_j\left(\int_0^t \alpha_j(\mathbf{X}(h))dh\right),$$

where Y_j are independent Poisson processes with intensity one. Hence, for $i \in \{1, \ldots, n\}$

$$X_i(t) = X_i(0) + \sum_{j=1}^{m} d_{ji} \cdot Y_j \left(\int_0^t \alpha_j(\mathbf{X}(h))dh \right), \tag{6.4}$$

where $\mathbf{d}_j = (d_{j1}, \ldots, d_{jn})$. The next step is to define $\mathbf{Z}(t) = \mathbf{X}(t) \cdot \mathcal{N}^{-1}$, that is, $\mathbf{Z}(t) = (Z_1(t), \ldots, Z_n(t))$ contains the concentrations of the chemical species in moles per liter at time t. Thus,

$$Z_i(t) = Z_i(0) + \sum_{j=1}^{m} d_{ji} \cdot \mathcal{N}^{-1} \cdot Y_j \left(\int_0^t \alpha_j(\mathbf{X}(h))dh \right), \tag{6.5}$$

and using the fact that $\alpha_j(\mathbf{s}) \approx \mathcal{N} \cdot \beta_j(\mathbf{z})$ yields

$$Z_i(t) \approx Z_i(0) + \sum_{j=1}^{m} d_{ji} \cdot \mathcal{N}^{-1} \cdot Y_j \left(\mathcal{N} \cdot \int_0^t \beta_j(\mathbf{Z}(h))dh \right). \tag{6.6}$$

By the law of large numbers, the unit Poisson process Y_j will approach $\mathcal{N} \cdot u$ at time $\mathcal{N} \cdot u$ for large $\mathcal{N} \cdot u$. Thus, $Y_j(\mathcal{N} \cdot u) \approx \mathcal{N} \cdot u$ and hence,

$$Z_i(t) \approx Z_i(0) + \sum_{j=1}^{m} d_{ji} \cdot \int_0^t \beta_j(\mathbf{Z}(h))dh. \tag{6.7}$$

The right-hand side of the above integral equation is the solution $\mathbf{z}(t)$ of the system of ODEs

$$\frac{d\mathbf{z}(t)}{dt} = \sum_{j=1}^{m} \mathbf{d}_j \cdot \beta_j(\mathbf{z}(t)). \tag{6.8}$$

As shown by Kurtz [70], in the large volume limit, where the volume and the number of molecules approach infinity (while the concentrations remain constant), $\mathbf{Z}(t) \to \mathbf{z}(t)$ in probability for finite times t. Note that the chemical concentrations $\mathbf{z}(t)$ evolve continuously and deterministically in time.

This continuous deterministic approximation is reasonable if all species have a small relative variance and if they are only weakly correlated. The reason is that only in this case the assumption that $Z_i(t)$ is deterministic is appropriate. Note that for most models this is the case if the population of species i is large since this implies that $E[X_i(t)]$ is large whereas the occurrence of chemical reactions results only in a marginal relative change of the value of $X_i(t)$.

The scaled solution of the differential equation (6.8)

$$\frac{d\mathbf{x}(t)}{dt} = \sum_{j=1}^{m} \mathbf{d}_j \cdot \alpha_j(\mathbf{x}(t)). \tag{6.9}$$

with initial condition $\mathbf{x}(0) = E[\mathbf{X}(0)]$ converges in probability to the scaled process $\mathbf{X}(t)$ for finite times t [70]. We call this equation the *reaction rate equation* (RRE). This result holds in the large volume limit, i.e., where the volume and the number of molecules approach infinity (while the concentrations remain constant). Note that the values $\mathbf{x}(t)$ evolve continuously and deterministically in time.

Example 6.1. *The ODEs of the exclusive switch (see Example 8.1) are given by*

$$\begin{aligned}
\frac{dz_1(t)}{dt} &= k_1 \cdot z_3(t) - k_3 \cdot z_1(t) - k_5 \cdot z_1(t) \cdot z_3(t) \\
&\quad + k_7 \cdot z_4(t) + k_9 \cdot z_4(t) \\
\frac{dz_2(t)}{dt} &= k_2 \cdot z_3(t) - k_4 \cdot z_2(t) - k_6 \cdot z_2(t) \cdot z_3(t) \\
&\quad + k_8 \cdot z_5(t) + k_{10} \cdot z_5(t) \\
\frac{dz_3(t)}{dt} &= -k_5 \cdot z_1(t) \cdot z_3(t) - k_6 \cdot z_2(t) \cdot z_3(t) \\
&\quad + k_7 \cdot z_4(t) + k_8 \cdot z_5(t) \\
\frac{dz_4(t)}{dt} &= k_5 \cdot z_1(t) \cdot z_3(t) - k_7 \cdot z_4(t) \\
\frac{dz_5(t)}{dt} &= k_6 \cdot z_2(t) \cdot z_3(t) - k_8 \cdot z_5(t)
\end{aligned}$$

where $z_1(t), z_2(t), z_3(t), z_4(t), z_5(t)$ denote the respective chemical concentrations. Moreover, for the stochastic constants c_j, as defined above, we have $c_j = \mathcal{N}^{-1} \cdot k_j$ for $j \in \{5, 6\}$ and $c_j = k_j$ for $j \notin \{5, 6\}$.

6.2.2 Propagation Model

For a given TCM $M = \langle \mathcal{S}, \mathbf{y}, \{\mathcal{C}_1, \ldots \mathcal{C}_m\} \rangle$, we construct the propagation model $N = \langle \{a\}, \mathcal{M}, \zeta, \pi \rangle$, where:

– the state space has a single state: a,

– the mass space $\mathcal{M} = \mathbb{R}^n_{\geq 0}$,

– the initial mass vector $\zeta_a = \mathbf{y}$ takes the value of the initial state of M,

– the edge function $\pi_{a\to a}(\mu) = \sum_{j=1}^{m} \mathbf{d}_j \cdot \alpha_j(\mu)$.

From the definition of the continuous-time propagation process of N, and from Equation (6.9) it follows directly that

$$g_a^{(t)} = \mathbf{x}(t).$$

We note that this PM is not conservative, as it propagates mass on a self loop.

Chapter 7

Naive Hybrid Numerical Solution

Assume that we have a system where certain species have a large population. We propose to approximate those populations with continuous deterministic variables and to keep the remaining population variables discrete stochastic. This is motivated by the fact that it is usually infeasible or at least computationally very costly to solve a purely stochastic model with high populations since in the respective dimensions the number of significant states is large. In this chapter we therefore switch to a hybrid model where the stochastic part does not contain large populations and the deterministic part monitors the average behaviour of large populations. This way we obtain a fast approximation of the solution. However, the approximation proves not to be accurate for all systems, problem with which we will deal in the next chapter where we propose a more refined hybrid stochastic model.

7.1 Naive Species Aggregation

For a TCM M, let $D = [1 \dots n]$ be the species index set, and let $\left\{\hat{D}, \bar{D}\right\}$ be a split of D into species with a small population and species with a large population, respectively. We also define $\hat{n}$ to be the size of $\hat{D}$, and $\bar{n}$ to be the size of $\bar{D}$, such that $n = \hat{n} + \bar{n}$. For now, we consider this partition to be static and in Section 7.4 we address the problem of dynamically switching species from one category to another one.

Definition 7.0.1. *($\bar{D}$-Partition) For a split $\left\{\hat{D}, \bar{D}\right\}$ of the species, we define the* $\bar{D}$-partition $\hat{\mathcal{S}}$ *to be the space $\mathcal{S}$ where the $\bar{D}$ components are projected away:*

$$\hat{\mathcal{S}} = \left\{\mathbf{s} \in \mathbb{N}_{0,\top}^{n} \mid (s_i \in \mathbb{N}_0,\ \forall i \in \hat{D}) \text{ and } (s_i = \top,\ \forall i \in \bar{D})\right\},$$

with $\mathbb{N}_{0,\top}$ *the set of natural numbers extended with* 0 *and with the symbol* $\top$*. Through an abuse of notation, we let each element of* $\hat{\mathcal{S}}$ *to represent a subset of* $\mathcal{S}$*, that is, for all* $A \in \hat{\mathcal{S}}$ *we have that:*

$$A = \left\{ \mathbf{s} \in \mathcal{S} \mid A_i = s_i,\ \forall i \in \hat{D} \right\}.$$

We refer to the elements of $\bar{D}$ *as aggregated components or species and to the elements of* $\hat{\mathcal{S}}$ *as aggregated states. It follows that* $\hat{\mathcal{S}}$ *defines a partition of* $\mathcal{S}$ *and the space* $\hat{\mathcal{S}}$ *is called an* aggregated space*.*

We note by $\hat{\mathbf{s}}$ and $\bar{\mathbf{s}}$ the projections of $\mathbf{s}$ on one set of species or another, and we use the same notations for the rate functions α, the update functions u and the change vectors $\mathbf{d}$ of a TCM.

Definition 7.0.2 (Naive $\bar{D}$-aggregated solution)**.** *Let* $\mathbf{p}^{(t)}$ *be the transient solution at time* t *of a TCM* M*. The* naive $\bar{D}$-aggregated solution *of* M *at time* t*, with respect to the aggregated species in* $\bar{D}$*, is a tuple* $\left\langle \hat{\mathbf{p}}^{(t)}, \bar{\boldsymbol{\mu}}^{(t)} \right\rangle$*, such that*

– $\hat{\mathbf{p}}^{(t)} \in \left[\hat{\mathcal{S}} \to [0,1] \right]$ *is a vector of aggregated probabilities with entries:*

$$\hat{p}_A^{(t)} = \sum_{\mathbf{s} \in A} p_{\mathbf{s}}^{(t)}, A \in \hat{\mathcal{S}},$$

– $\bar{\boldsymbol{\mu}}^{(t)} \in \mathbb{R}^{\bar{n}}$ *is the expectation of the aggregated components* $\bar{D}$*:*

$$\bar{\boldsymbol{\mu}}^{(t)} = \sum_{\mathbf{s} \in \mathcal{S}} p_{\mathbf{s}}^{(t)} \cdot \bar{\mathbf{s}},$$

where $\bar{\mathbf{s}}$ *is the* $\bar{D}$ *projection of the state* $\mathbf{s}$*.*

7.2 Mathematical Model

An exact computation of the $\bar{D}$-aggregated solution, as defined above, can only be done indirectly by first computing the transient solution of the TCM M at time t, and then aggregating it.

Here we propose a mathematical model that approximates the naive $\bar{D}$-aggregated solution with the solution of a stochastic process $\left(\tilde{\mathbf{X}}(t)\right)_{t \geq 0}$ defined hereafter. For this model we then construct an equivalent propagation model $\tilde{N}$, from which we can compute a solution directly from the much smaller aggregated state space, trading accuracy for performance.

The process $\tilde{\mathbf{X}}(t)$ has n components that we split with respect to the species partition $\left\{\hat{D}, \bar{D}\right\}$. To ease the notation, let $\boldsymbol{V}(t) = \tilde{\mathbf{X}}_{\hat{D}}(t)$, and let $\mathbf{W}(t) = \tilde{\mathbf{X}}_{\bar{D}}(t)$, in other words, $\boldsymbol{V}(t)$ are the random variables that represent small populations, while the random variables in $\mathbf{W}(t)$ represent large populations.

We define $\mathbf{W}(t)$ deterministically, as the solution of the following ODE

$$\frac{d\mathbf{W}(t)}{dt} = \sum_{j=1}^{m} \bar{\mathbf{d}}_j \cdot E[\alpha_j(\boldsymbol{V}(t), \mathbf{W}(t))]. \tag{7.1}$$

Note that this equation is a direct consequence of Equation (6.1) if we assume that the populations $\mathbf{W}(t)$ evolve deterministically and continuously. We initialize the values $\mathbf{W}(0)$ with the respective expected populations at time zero.

We handle the discrete stochastic variables as follows. The probability distribution $\hat{p}^{(t)}_{\mathbf{v}} = Pr(\boldsymbol{V}(t) = \mathbf{v})$ of the random vector $\boldsymbol{V}(t)$ is given by the CME:

$$\begin{aligned}\frac{d\hat{p}^{(t)}_{\mathbf{v}}}{dt} =& \sum_{j=1}^{m} \Big(\alpha_j(\mathbf{v} - \hat{\mathbf{d}}_j, \mathbf{W}(t)) \cdot \hat{p}^{(t)}(\mathbf{v} - \hat{\mathbf{d}}_j) \\ & - \alpha_j(\mathbf{v}, \mathbf{W}(t)) \cdot \hat{p}^{(t)}_{\mathbf{v}}\Big)\end{aligned} \tag{7.2}$$

which operates on a reduced state space $\hat{\mathcal{S}}$ of dimension $\hat{n}$. The propensity functions α_j are defined as before but the original argument is now split into the two parts, namely the $\hat{n}$-dimensional state $\mathbf{v}$ and the $\bar{n}$-dimensional real-valued vector $\mathbf{W}(t)$ of the populations that are approximated by the continuous deterministic functions as defined in Equation (7.1). Note that Equation (7.1) depends on Equation (7.2) because the expectation $E[\alpha_j(\boldsymbol{V}(t), \mathbf{W}(t))]$ is computed as

$$E[\alpha_j(\boldsymbol{V}(t), \mathbf{W}(t))] = \sum_{\mathbf{v}} \hat{p}^{(t)}_{\mathbf{v}} \cdot \alpha_j(\mathbf{v}, \mathbf{W}(t)).$$

Example 7.1 (Crystallization)**.** *In Example 4.4 we treat species A deterministically. Since in this case the number of reactions of type $A + A \to B$ until time t is deterministic, we treat the population of B as a deterministic variable as well. Thus, $\tilde{\mathbf{X}}(t) = (W_1(t), W_2(t), V_3(t), V_4(t))$. We update $W_1(t)$ and $W_2(t)$ according to the ODEs*

$$\begin{aligned}\frac{d}{dt}W_1(t) =& - k_1 \cdot W_1(t) \cdot W_1(t) - k_2 \cdot W_1(t) \cdot E[V_3(t)] \\ \frac{d}{dt}W_2(t) =& k_1 \cdot W_1(t) \cdot W_1(t).\end{aligned}$$

The probability distribution of the remaining populations $V_3(t)$ and $V_4(t)$ is given by

$$\frac{d\hat{p}_{\mathbf{v}}^{(t)}}{dt} = \alpha_2(\mathbf{v} - \hat{\mathbf{d}}_2, \mathbf{W}(t)) \cdot \hat{p}^{(t)}(\mathbf{v} - \hat{\mathbf{d}}_2) - \alpha_2(\mathbf{v}, \mathbf{W}(t)) \cdot \hat{p}_{\mathbf{v}}^{(t)}$$
$$= c_2 \cdot (v_3 + 1) \cdot w_1(t) \cdot \hat{p}^{(t)}((v_3 + 1, v_4 - 1)) - c_2 \cdot v_3 \cdot w_1(t) \cdot \hat{p}^{(t)}((v_3, v_4)),$$

where $\mathbf{v} = (v_3, v_4)$ is a state in the stochastic process $(\boldsymbol{V}(t))_{t \geq 0}$ with $\boldsymbol{V}(t) = (V_3(t), V_4(t))$.

Equation (7.1) and (7.2) suggest a numerical integration procedure for the approximation of $\mathbf{W}(t)$ as well as the distribution of $\boldsymbol{V}(t)$ as described in the sequel. But first let's define a propagation model for the transient analysis of $\tilde{X}(t)$.

7.3 Propagation Model

We construct a propagation model for the transient solution of the stochastic process that we have just defined in Section 7.2.

The definition of a propagation model is such that the rate of a reaction depends on the state from which the reaction triggers and on the mass that state holds at the current time. Here, we want to make the rate dependent on the overall expectation of the population of the aggregated species. We are therefore obliged to copy this information in the mass of each state. Note, however, that the real implementation is not doing this, but it is keeping the duplicated information in a single variable.

We define the propagation model $\tilde{N} = \left\langle \hat{\mathcal{S}}, \mathcal{M}, \zeta, \pi, \tau \right\rangle$ as follows:

– the mass space $\mathcal{M} = [0, 1] \times \mathbb{R}_{\geq 0}^{\bar{n}}$, allows the propagation of probabilities and the replication of the information about the expectation of the aggregated species $\bar{D}$,

– the initial vector ζ with elements

$$\zeta_A = \begin{cases} \langle 1, \hat{\mathbf{y}} \rangle & \text{if } \mathbf{y} \in A, \\ \langle 0, \mathbf{0} \rangle & \text{if } \mathbf{y} \notin A, \end{cases}$$

– the edge function π is defined by its components (π^p, π^μ), where:

Algorithm 7.1 The main loop of the algorithm.

Input: TCM $M = \langle \mathcal{S}, \mathbf{y}, \{\mathcal{C}_1, \ldots \mathcal{C}_m\}\rangle$, time horizon T, threshold δ, boundary $\boldsymbol{b}$;
Output: $\hat{\mathbf{p}}(t)$ and $W(t)$;
Variables: $t = 0$; $\hat{D} = \{1, \ldots, n\}$;$\bar{D} = \{\}$;$\mathbf{S} = \{\mathbf{y}\}$;

1: **while** $t < T$ **do**
2: *check_stoch_vars*; // stoch. $\rightsquigarrow$ determ.
3: *check_det_vars*; // determ. $\rightsquigarrow$ stoch.
4: $\Delta \leftarrow \min_{\mathbf{x}\in\mathbf{S}} \frac{1}{\lambda(\mathbf{x})}$; // time step
5: *update_det_vars*; // see Equation (7.1)
6: *update_stoch_vars*; // see Equation (7.2)
7: $t \leftarrow t + \Delta$;

the probability edge function π^p is

$$\pi^p_{A\to A+\hat{\mathbf{d}}_j}(\langle p, \mu\rangle) = p \cdot \alpha_j(A, \mu), \forall A \in \hat{\mathcal{S}}, j = 1 \ldots m,$$

and the expectation edge function π^μ is

$$\pi^\mu_{A\to A}(\langle p, \mu\rangle) \quad = \sum_{j=1}^{m} \bar{\mathbf{d}}_j \cdot \alpha_j(A, \mu), \forall A \in \hat{\mathcal{S}}.$$

The continuous-time propagation process $\tilde{\mathbf{g}}^{(t)}$ of the propagation model $\tilde{N}$ that we have just defined is approximating the naive $\bar{D}$-aggregated solution of the TCM M because, from the definition of $\tilde{N}$, the valuation of $\tilde{\mathbf{g}}^{(t)}$ equals $\langle Pr(\boldsymbol{V}(t) = \cdot), W(t)\rangle$, where $Pr(\boldsymbol{V}(t) = \cdot)$ is a probability vector with entries $Pr(\boldsymbol{V}(t) = \mathbf{v})$. We can use one of the propagation algorithms that we have already presented to solve $\tilde{\mathbf{g}}^{(t)}$, and in the next section we address the case where we use the forth order Runge-Kutta method (RK4) for this problem.

7.4 Numerical Approximation Algorithm

Given a Markov chain $(\mathbf{X}(t))_{t\geq 0}$ that describes a chemical reaction network, we approximate the naive aggregated solution at time T as outlined in Algorithm 7.1. The data structure $\mathbf{S}$ contains a node $\mathbf{x}$ for each state $\mathbf{v}$ of $V(t)$ of the computed solution. The value $\mathbf{x}.\mathbf{v}$ refers the value of the state, and $\mathbf{x}.p$ to the probability hold by the state. The variable vector $\mathbf{w}$, with entries $w_1 \ldots w_n$, holds the expectation of $\mathbf{W}(t)$ at the current time.

We start with $t = 0$ and define the set $\hat{D} = D$, that is, we treat all species as discrete stochastic. The set of indices of deterministic variables is thus empty, $\bar{D} = \{\}$. In line 2 of Algorithm 7.1, we call the function *check_stoch_vars*

Algorithm 7.2 *check_stoch_vars*

Input: species split $\hat{D}, \bar{D}$, space structure $\mathbf{S}$
Output: species split $\hat{D}, \bar{D}$, space structure $\mathbf{S}$
Variables: $\bar{D}' \leftarrow \{\}$;
1: **for all** $i \in \hat{D}$ **do**
2: $E_i \leftarrow \sum_{\mathbf{x}.\mathbf{v} \in \mathbf{S}} \mathbf{x}.v_i \cdot \mathbf{x}.p$; // Compute the expectation of $V_i(t)$
3: **if** $E_i \geq b_i$ **then**
4: $\bar{D}' \leftarrow \bar{D}' \cup \{i\}$;
5: $w_i \leftarrow E_i$;
6: **if** $\bar{D}' \neq \{\}$ **then**
7: $\bar{D} \leftarrow \bar{D} \cup \bar{D}'$; $\hat{D} \leftarrow \hat{D} \setminus \bar{D}'$;
8: // compute marginal distribution
9: $\mathbf{S}' \leftarrow \{\}$; // Construct a new structure $\mathbf{S}$
10: **for all** $\mathbf{x} \in \mathbf{S}$ **do**
11: $\mathbf{x}' \leftarrow find(S', \mathbf{x}.\mathbf{v}_{|\hat{D}})$;
12: **if** $\mathbf{x}' = null$ **then**
13: // Create new aggregated state.
14: $\mathbf{x}'.\mathbf{v} \leftarrow \mathbf{x}.\mathbf{v}_{|\hat{D}}$;
15: $\mathbf{x}'.p \leftarrow 0$;
16: $\mathbf{S}' \leftarrow \mathbf{S}' \cup \{\mathbf{x}'\}$;
17: $\mathbf{x}'.p \leftarrow x'.p + x.p$;
18: $\mathbf{S} \leftarrow \mathbf{S}'$;

to determine whether stochastic variables are transformed into deterministic variables. The function *check_det_vars* (line 3) is called in each iteration in order to check whether deterministic variables are transformed into stochastic variables.

Switching from Stochastic to Deterministic and Vice Versa. Algorithm 7.2 gives the description of the function *check_stoch_vars*. If $E[V_i(t)]$ is larger than the threshold b_i we remove index i from $\hat{D}$ and add it to the set $\bar{D}$ (line 7). We then set the variable $w_i = E[V_i(t)]$ (line 5), thus replacing $V_i(t)$ by the deterministic variable $W_i(t)$ that evolves according to Equation (7.1).

Also, when switching some species to deterministic mode we need to compute the marginal distribution of the remaining random variables in order to reduce the dimension of the state space from $\hat{D}$ to $\hat{D} \setminus \bar{D}'$ (lines 10-18). When dimensions $\bar{D}'$ are switched from a stochastic to a deterministic representation, all states that are equal on the $\hat{D} \setminus \bar{D}'$ dimensions are aggregated into a single state that accumulates the sum of their probabilities. Assume that $\mathbf{x}'$ is such a new node that aggregates several states, then we set

$$\mathbf{x}'.p = \sum_{\substack{\mathbf{x} \in \mathbf{S}: \\ \mathbf{x}.\mathbf{v}_{|\hat{D} \setminus \bar{D}'} = \mathbf{x}'.\mathbf{v}}} p_{\mathbf{v}}^{(t)}.$$

Algorithm 7.3 *check_det_vars*

Input: species split $\hat{D}, \bar{D}$, space structure $\mathbf{S}$
Output: species split $\hat{D}, \bar{D}$, space structure $\mathbf{S}$
Variables: $\hat{D}' \leftarrow \{\}$;
1: **for all** $i \in \bar{D}$ **do**
2: // check population bound
3: **if** $w_i < b_i$ **then**
4: $\hat{D}' \leftarrow \hat{D}' \cup \{i\}$;
5: **if** $\hat{D}' \neq \{\}$ **then**
6: $\hat{D} \leftarrow \hat{D} \cup \hat{D}'$; $\bar{D} \leftarrow \bar{D} \setminus \hat{D}'$;
7: // add dimensions to the stochastic model
8: **for all** nodes $\mathbf{x} \in \mathbf{S}$ **do**
9: **for all** $i \in \bar{D}'$ **do**
10: add i-th entry w_i to $\mathbf{x}.\mathbf{v}$

Note that *check_stoch_vars* is implemented efficiently by using an appropriate hash function.

In a similar way, we switch back to a discrete stochastic treatment of species i using the function *check_det_vars* (see Algorithm 7.3). Switching from deterministic to stochastic requires to add a dimension to the discrete stochastic part of the model. Thus, we add an entry $\mathbf{x}.v_i = w_i$ to all states in the current set of significant states (compare line 8-10 in Algorithm 7.3). After this operation, we have the equivalent of $Pr(V_i(t) = w_i) = 1$.

Mutual Update of Stochastic and Deterministic Variables. We use Equations (7.1) and (7.2) to approximate the future behaviour of the system. Algorithm 7.1 illustrates the case where the continuous-time propagation model is solved using a fourth order Runge-Kutta method (see Section 4.4) with an integration step of Δ, where we first update the deterministic variables and then the stochastic variables. The step Δ is chosen such that during $[t, t + \Delta)$ the distribution of the discrete stochastic variables do not change significantly. In our implementation we obtained the best results if Δ equals the shortest average residence time of a state $\mathbf{v}$ in $\mathbf{S}$ (see line 4 in Algorithm 7.1). Note that the average residence time in state $\mathbf{v}$ is computed as $1/\lambda(\mathbf{x})$, where $\lambda(\mathbf{x}) = \sum_{j=1}^{m} \alpha_j(\mathbf{x}.\mathbf{v})$.

7.5 Experimental Results

We implemented the algorithm described in Section 7.4 in C++ and run experiments on an Intel 3GHz Linux workstation with 3GB of RAM. In this section we present three examples one which we applied our algorithm. All three were

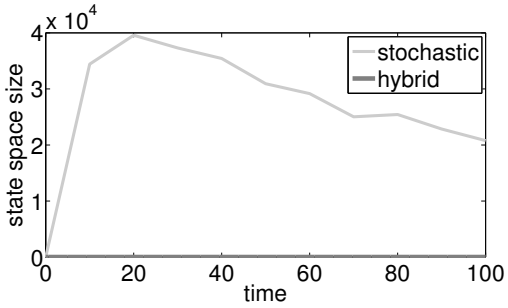


Figure 7.1: Size of the state space of the purely stochastic model and the hybrid model.

chosen such that a full stochastic solution is possible in order to compare it to the solution of the stochastic hybrid model.

The first example is the crystallization presented in Example 4.4. The second example is a model for the transcription regulation of a repressor protein in bacteriophage λ [44]. This protein is responsible for maintaining lysogeny of the λ virus in E. coli. It has 6 different chemical species and 10 reactions. We vary the parameters of this example in order to increase/decrease the populations of certain chemical species. In the third example the two involved populations show sustainable periodic oscillations. We use it to test the switching mechanism of our algorithm.

Crystallization. For the simple crystallization in Example 4.4 we used the parameters from [50]. Since the initial population of A is $w_1(0) = 10^6$ and that of C is $\hat{X}_3(0) = 10$ we treated A deterministic and C stochastic. Thus, the number of reactions of type $A + A \to B$ is deterministic continuous, which implies that the population $w_2(t)$ of B is deterministic continuous as well. Similarly, the population $\hat{X}_4(t)$ of molecules of type D is represented by a stochastic discrete variable because of $A + C \to D$.

We solved all three of models of the crystallization. The purely stochastic model, the purely deterministic model, and the hybrid model. The running times for a time horizon of $T = 100$ are 6 min 18 sec (purely stochastic), 6 sec (hybrid) and 3 sec (deterministic). We compared the solution of the hybrid model to that of the stochastic model by considering the first ten moments of the distribution of $\hat{X}_3(t)$ and $\hat{X}_4(t)$. We found that the relative errors of the first six moments of the random variable $\hat{X}_3(t)$ are all of the order 10^{-5}, where we averaged over time. Higher moments were of the order of 10^{-4}. We also compared the value of $w_1(t)$ to the expectation of A in the purely stochastic model and got a relative error of $2 \cdot 10^{-4}$. Figure 7.1 shows the size of the sets $\mathbf{S}$ over time in the stochastic model and in the hybrid model. In the former

pset	running times			$\lvert\mathbf{S}\rvert$	
	ODE	hybrid	stoch	hybrid	stoch
1	<1sec	17min	65min	$1 \cdot 10^4$	$8 \cdot 10^6$
2	<1sec	13min	163min	$9 \cdot 10^3$	$2 \cdot 10^6$

Table 7.1: Running times of the phage λ model.

model, the size goes up to $4 \cdot 10^4$ whereas in the hybrid model, there are at most 11 states.

Goutsias Model. For Example 4.2 we used two parameter sets (pset 1 and pset 2 in Table 7.1) that both differ from the original parameters that we used in previous work [56] in that one increases the number of RNA molecules and one increases the number of M molecules. The first one uses constants $c_1 = 0.043$, $c_2 = 7 \cdot 10^{-4}$, $c_3 = 71.5$, $c_4 = 3.9 \cdot 10^{-6}$, $c_5 = 0.02$, $c_6 = 0.48$, $c_7 = 2 \cdot 10^{-4}$, $c_8 = 0.9 \cdot 10^{-11}$, $c_9 = 0.08$, $c_{10} = 0.5$. The second one uses constants $c_1 = 4.3$, $c_2 = 7 \cdot 10^{-5}$, $c_3 = 0.07$, $c_4 = 3.9 \cdot 10^{-4}$, $c_5 = 0.2$, $c_6 = 0.048$, $c_7 = 2 \cdot 10^{-5}$, $c_8 = 0.9 \cdot 10^{-12}$, $c_9 = 8 \cdot 10^{-4}$, $c_{10} = 5$. We always start initially in state $(10, 10, 10, 2, 0, 0)$ and choose the time horizon as the latest time point where we are able to do a purely stochastic solution. For pset 1, we run out of memory at time $T = 6$ and for pset 2 we time out at time $T = 22$, because computing time $T = 23$ would take more than 3 hours.

Table 7.1 shows running times for purely stochastic model (stoch), the purely deterministic model (ODE), and the hybrid model (hybrid). Note that the hybrid solution as well as the deterministic solution are feasible for much longer time horizons. In the columns with label $|\mathbf{S}|$ we list the maximal size of the set $\mathbf{S}$ during the hybrid and the stochastic solution. The memory requirements are proportional to this number. We compared the distributions obtained from the hybrid solution with the distributions of the purely stochastic model. At the final time instant, we get for the first four moments relative errors of 0.02, 0.10, 0.20, 0.29 (pset 1) and 0.02, 0.04, 0.05, 0.07 (pset 2) where we averaged over all species that were kept stochastic in the hybrid model. We switched to a deterministic treatment at a population threshold of 50. Higher population thresholds yield more accurate solutions but require more computation time. The increase of the size of the set of significant states makes the purely stochastic solution infeasible for longer time horizons. As opposed to that the memory requirements of the hybrid solution are tractable.

Predator Prey. We apply our algorithm to the predator prey model described in [38]. It involves two species A and B and the reactions are $A \to 2A$, $A + B \to 2B$, and $B \to \emptyset$. The model shows sustainable periodic oscillations until eventually the population of B reaches zero. We choose rate constants $c_1 = 1$, $c_2 = 0.01$, $c_3 = 1$ and start with the initial values 200 for A and 100 for B. It is important to note that for this example both the purely stochastic solution as well as the ODE solution are difficult, because eventually the population of B is zero and then the population of A increases exponentially. In this case, the time step of Monte Carlo simulation methods becomes smaller and smaller, which makes the generation of trajectories very time consuming. Similarly, the fast RK4 algorithm that we use for a purely stochastic solution becomes slow and uses large amounts of memory. The ODE solution, on the other hand, can accurately approximate the fast growth of A but it fails to detect the extinction of B. The hybrid model that we propose is able to handle both the exponential growth of A if B is small and the probability of extinction of B. The reason is that if A becomes large, then we represent it as a continuous deterministic variable. We used a population threshold of 150. The full stochastic solution times out after 200 min, which is the moment at which it has solved a time horizon of 9. The running time of the hybrid solution is 14 min and the running time of the purely stochastic solution is 56 min. The relative error of the first two moments at the final time are 0.04, 0.35. Note that the full stochastic solution has large memory requirements for later time instances whereas the memory requirements of the hybrid solution remain tractable.

Chapter 8

Hybrid Numerical Solution

In this chapter we refine the stochastic hybrid approach presented in Chapter 7 to more accurately estimate the solution of systems containing both small and large populations. More precisely, we maintain the discrete stochastic representation for small populations, and we associate with each state of the aggregated stochastic process a continuous deterministic variable that represents the expectation of the large populations *conditioned* on the value of the aggregated state.

We compute the solution of a CME with a reduced dimension as well as the solution of a system of (non-linear) ordinary differential equations. The former describes the distribution of the discrete stochastic variables and the latter the values of the continuous deterministic variables, and the two descriptions depend on each other. Assume, for instance, that a system has two chemical species. The two population sizes at time t are represented by the random variables $W(t)$ and $V(t)$, where $W(t)$ is large and $V(t)$ is small. Then, we consider for $V(t)$ all events $V(t) = v$ that have significant probability, i.e., $Pr(V(t) = \mathbf{y})$ is greater than a certain threshold. For $W(t)$ we consider the conditional expectations $E[W(t) \,|\, V(t) = v]$ and assume that they change continuously and deterministically in time. We iterate over small time steps $dt > 0$ and, given the distribution for $V(t)$ and the values $E[W(t) \,|\, V(t) = v]$, we compute the distribution of $V(t + dt)$ and the values $E[W(t + dt) \,|\, V(t + dt) = v]$. Again, we restrict our computation to those values of v that have significant probability.

8.1 Conditional Species Aggregation

As in Chapter 7, we split the set of species indexes D into small populations $\hat{D}$ and large populations $\bar{D}$. Recall that $\hat{\mathcal{S}}$ denotes the space aggregated with respect to species $\bar{D}$, and that for all $s \in \mathcal{S}$ the aggregated state $\hat{\mathbf{s}}$ belongs to $\hat{\mathcal{S}}$ and is a subset of $\mathcal{S}$.

Definition 8.0.3 (Conditional $\bar{D}$-aggregated solution)**.** *Let $\boldsymbol{p}^{(t)}$ be the transient solution at time t of a propagation model M. The* conditional $\bar{D}$-aggregated solution *of M, at time t, with respect to the aggregated species $\bar{D}$, is a tuple $\left\langle \hat{\mathbf{p}}^{(t)}, \bar{\boldsymbol{\mu}}^{(t)} \right\rangle$, where:*

$\hat{\mathbf{p}}^{(t)}$ is a vector of probabilities over the aggregated state space: $\hat{\mathbf{p}}^{(t)} \in \left[\hat{\mathcal{S}} \to [0,1]\right]$, with elements:

$$\hat{p}_A^{(t)} = \sum_{\mathbf{s} \in A} p_{\mathbf{s}}^{(t)},$$

and $\bar{\boldsymbol{\mu}}^{(t)}$ is a vector of conditional expectations over the aggregated state space: $\hat{\mu}^{(t)} \in \left[\hat{\mathcal{S}} \to \mathbb{R}^n\right]$ with elements:

$$\bar{\mu}_A^{(t)} = \sum_{\mathbf{s} \in A} p_{\mathbf{s}}^{(t)} \cdot \bar{\mathbf{s}},$$

where $\bar{s}$ is the $\bar{D}$ projection of $\mathbf{s}$.

We observe that $\bar{\boldsymbol{\mu}}^{(t)}$ is now a vector of conditional expectations over $\hat{\mathcal{S}}$, and not a global expectation as in the naive definition given in Section 7.1.

8.2 Mathematical Model

As in the case of the naive hybrid stochastic solution, a direct method for computing the conditional $\bar{D}$-aggregated solution does not exist and one must rely on approximation in order to take advantage of the smaller size of the aggregated state space $\hat{\mathcal{S}}$.

Here we propose a mathematical model that approximates the conditional $\bar{D}$-aggregated solution with the solution of a stochastic process $(\tilde{x}(t)(t))_{t \geq 0}$ defined as follows.

The stochastic process $\tilde{x}(t)$ is a set of n random variables that we split with respect to the species split $\left\{\hat{D}, \bar{D}\right\}$. Let $\boldsymbol{V}(t) = \tilde{x}_{\hat{D}}(t)$, and let $\mathbf{W}(t) = \tilde{x}_{\bar{D}}(t)$,

in other words, $\boldsymbol{V}(t)$ are the random variables that represent small populations, while $\mathbf{W}(t)$ are the random variables that represent large populations.

We assume that the evolution of $\mathbf{W}(t)$ is given by the stochastic differential equation

$$\frac{d\mathbf{W}(t)}{dt} = \sum_{j=1}^{m} \bar{\mathbf{d}}_j \cdot \alpha_j(\boldsymbol{V}(t), \mathbf{W}(t)). \tag{8.1}$$

Thus, if $\boldsymbol{V}(t)$ would be constant from time t on, with $\boldsymbol{V}(t) = \mathbf{v}$then the change of $\mathbf{W}(t)$ would be constant over time as in the reaction rate equation (6.9), and for an infinitesimal time length dt we have:

$$\mathbf{W}(t + dt) = \mathbf{W}(t) + \sum_{j=1}^{m} \bar{\mathbf{d}}_j \cdot \alpha_j(\mathbf{v}, \mathbf{W}(t)) \cdot dt. \tag{8.2}$$

The evolution of $\boldsymbol{V}(t)$ remains as defined for the Markov chain, i.e.,

$$Pr\Big(\boldsymbol{V}(t + dt) = \mathbf{v} + \hat{\mathbf{d}}_j \mid \boldsymbol{V}(t) = \mathbf{v}, \mathbf{W}(t) = \mathbf{w}\Big) = \alpha_j(\mathbf{v}, \mathbf{w}) \cdot dt,$$

for an infinitesimal time length dt.

The density function $h^{(t)}(\mathbf{v}, \mathbf{w})$ of the Markov process $\{(\boldsymbol{V}(t), \mathbf{W}(t)), t \geq 0\}$ can be derived in the same way as done by Horton et al. [63]. Here, for simplicity we consider only the case $\bar{n} = 1$ which means that $\mathbf{w} = w$ is a scalar. The generalization to higher dimensions is straightforward. If $w > 0$ then the following partial differential equation holds for h.

$$\begin{aligned}&\frac{\partial h^{(t)}(\mathbf{v}, w)}{\partial t} + \frac{\partial\big(h^{(t)}(\mathbf{v}, w) \cdot \sum_{j=1}^{m} \bar{\mathbf{d}}_j \cdot \alpha_j(\mathbf{v}, w)\big)}{\partial w} \\ =&\sum_j \alpha_j(\mathbf{v} - \hat{\mathbf{d}}_j, w) \cdot h^{(t)}(\mathbf{v} - \hat{\mathbf{d}}_j, w) - \sum_j \alpha_j(\mathbf{v}, w) \cdot h^{(t)}(\mathbf{v}, w).\end{aligned}$$

If $w = 0$ then we have probability mass $\rho^{(t)}(\mathbf{v}, w)$ in state $(\mathbf{v}, w)$ where

$$\begin{aligned}&\frac{\partial \rho^{(t)}(\mathbf{v}, w)}{\partial t} + h^{(t)}(\mathbf{v}, w) \cdot \sum_{j=1}^{m} \bar{\mathbf{d}}_j \cdot \alpha_j(\mathbf{v}, w) \\ =&\sum_j \alpha_j(\mathbf{v} - \hat{\mathbf{d}}_j, w) \cdot \rho^{(t)}(\mathbf{v} - \hat{\mathbf{d}}_j, w) - \sum_j \alpha_j(\mathbf{v}, w) \cdot \rho^{(t)}(\mathbf{v}, w).\end{aligned}$$

As explained in-depth by Horton et al., the above equations express that probability mass must be conserved, i.e. the change of probability mass in a "cell"

with boundaries $(\mathbf{v}, w-dw)$ and $(\mathbf{v}, w+dw)$ equals the total mass of probability entering the cell minus the total mass leaving the cell.

In order to exploit the fact that the relative variance of $\mathbf{W}(t)$ is small, we suggest an approximative solution of the stochastic hybrid model given above. The main idea is not to compute the full density h and the probability mass ρ but only the distribution of $\boldsymbol{V}(t)$ as well as the *conditional expectations* $E[\mathbf{W}(t) \,|\, \boldsymbol{V}(t) = \mathbf{v}]$. Thus, in our numerical procedure the distribution of $\mathbf{W}(t)$ is approximated by the different values $E[\mathbf{W}(t) \,|\, \boldsymbol{V}(t) = \mathbf{v}]$, $\mathbf{v} \in \hat{\mathcal{S}}$ that are taken by $\mathbf{W}(t)$ with probability $Pr(\boldsymbol{V}(t) = \mathbf{v})$.

Assume that at time t we have the approximation $\boldsymbol{p}^{(t)}$ of the discrete stochastic model, that is, for all states $\mathbf{s}$ that have a probability that is greater than δ we have $p_{\mathbf{s}}^{(t)} > 0$ and for all other states $\mathbf{s}$ we have $p_{\mathbf{s}}^{(t)} = 0$. At time t the expectations of one or more populations reached a certain large population threshold. Thus, we switch to a hybrid model where the large populations (index set $\bar{D}$) are represented as continuous deterministic variables $\mathbf{W}(t)$ while the small populations (index set $\hat{D}$) are represented by $\boldsymbol{V}(t)$. We first compute the vector of conditional expectations

$$\Psi_{\mathbf{v}}^{(t)} := E[\mathbf{W}(t) \,|\, \boldsymbol{V}(t) = \mathbf{v}] = \sum_{\mathbf{s} \in \mathbf{v}} \bar{\mathbf{s}} \cdot p_{\mathbf{s}}^{(t)},$$

and we recall that $\mathbf{v} \in \hat{\mathcal{S}}$ represents the set of states that are equal to $\mathbf{v}$ on dimensions $\hat{D}$. We also compute the distribution $\hat{\boldsymbol{p}}^{(t)}$ of $\boldsymbol{V}(t)$ as

$$\hat{p}_{\mathbf{v}}^{(t)} := \sum_{\mathbf{s} \in \mathbf{v}} p_{\mathbf{s}}^{(t)}.$$

Now, we integrate the system for a small time interval of length $dt > 0$. This is done in three steps as described below. For $t' \in [t, t+dt)$, we will write $\Psi_{\mathbf{v}}^{(t')}$ for the approximation of $E[\mathbf{W}(t') = \mathbf{w} \,|\, \boldsymbol{V}(t') = \mathbf{v}]$. The i-th element of the $\bar{n}$-dimensional vector $\Psi_{\mathbf{v}}^{(t')}$ is denoted by $\psi_{\mathbf{v},i}^{(t')}$. The value $\hat{p}_{\mathbf{v}}^{(t')}$ denotes the approximation of $Pr(\boldsymbol{V}(t') = \mathbf{v})$. The vector $\hat{\boldsymbol{p}}^{(t')}$ contains the elements $\hat{p}_{\mathbf{v}}^{(t')}$.

(1) Update distribution. We first integrate $\hat{\boldsymbol{p}}^{(t)}$ for dt time units according to a CME with dimension $\hat{n}$ to approximate the probabilities $Pr(\boldsymbol{V}(t+dt)=\mathbf{v})$ by $\hat{p}_{\mathbf{v}}^{(t+dt)}$, that is, $\hat{\boldsymbol{p}}^{(t+dt)}$ is the solution of the system of ODEs

$$\begin{aligned}\frac{d\hat{p}_{\mathbf{v}}^{(t')}}{dt'} =& \sum_{j=1}^{m} \alpha_j\left(\mathbf{v}-\hat{\mathbf{d}}_j, \Psi_{\mathbf{v}-\hat{\mathbf{d}}_j}^{(t')}\right)\cdot \hat{p}_{\mathbf{v}-\hat{\mathbf{d}}_j}^{(t')} \\ & - \sum_{j=1}^{m} \alpha_j(\mathbf{v}, \Psi_{\mathbf{v}}^{(t')})\cdot \hat{p}_{\mathbf{v}}^{(t')} \end{aligned} \tag{8.3}$$

with $t' \in [t, t+dt)$ and $\mathbf{v} \in \hat{\mathcal{S}}$. We use the initial condition $\hat{p}_{\mathbf{v}}^{(t)}$. Note that this equation is as Equation (2.4) except that the species in $\bar{D}$ are removed. Moreover, the population sizes $\mathbf{w}$ are replaced by the conditional expectations $\Psi_{\mathbf{v}}^{(t)}$ at time t. Note that we do not know the values $\Psi_{\mathbf{v}}^{(t')}$ for $t' > t$ and take $\Psi_{\mathbf{v}}^{(t)}$ as an approximation.

(2) Integrate. For each state $\mathbf{v} \in \hat{\mathcal{S}}$ with $\hat{p}_{\mathbf{v}}^{(t)} > \delta$, we compute an approximation $\Phi_{\mathbf{v}}^{(t+dt)}$ of the conditional expectation

$$E[\mathbf{W}(t+dt) \,|\, \boldsymbol{V}(t') = \mathbf{v}, t' \in [t, t+dt)],$$

that is, we assume that the system remains in state $\mathbf{v}$ during $[t, t+dt)$ and that the expected numbers of the large populations $\mathbf{W}(t)$ change deterministically and continuously in time. Thus, the $\bar{n}$-dimensional vector $\Phi_{\mathbf{v}}^{(t+dt)}$ is obtained by numerical integration of the ODE

$$\frac{d\Phi_{\mathbf{v}}^{(t')}}{dt'} = \sum_{j=1}^{m} \bar{\mathbf{d}}_j \cdot \alpha_j(\mathbf{v}, \Phi_{\mathbf{v}}^{(t')}) \tag{8.4}$$

with initial condition $\Phi_{\mathbf{v}}^{(t)} = \Psi_{\mathbf{v}}^{(t)}$. The above ODEs are similar to Equation (6.1) except that for $t' \in [t, t+dt)$ the value $E[\alpha_j(x(t'))]$ is approximated by $\alpha_j(\mathbf{v}, \Phi_{\mathbf{v}}^{(t')})$. For instance, if the j-th reaction is a bimolecular reaction that involves two populations with indices i, ℓ in $\bar{D}$ then the value $E[\alpha_j(\mathbf{v}, \mathbf{W}(t')) \,|\, \boldsymbol{V}(t') = \mathbf{v}]$ is approximated by $c_j \cdot \phi_{\mathbf{v},i}^{(t')} \cdot \phi_{\mathbf{v},\ell}^{(t')}$ where the two last factors are the elements of the vector $\Phi_{\mathbf{v}}^{(t')}$ corresponding to the i-th and ℓ-th population. Thus, in this case the correlations between the i-th and the ℓ-th populations are not taken into account which is reasonable if the two populations are large. Note that the correlations *are* taken into account when at least one population is represented as a discrete stochastic variable. If, for instance, $i \in \hat{D}$ and $\ell \in \bar{D}$, then we use the approximation $c_j \cdot v_i \cdot \Phi_{\mathbf{v},\ell}^{(t')}$ where v_i is the entry in vector $\mathbf{v}$ that represents the size of the i-th population.

(3) Distribute. In order to approximate $E[\mathbf{W}(t+dt) \,|\, \boldsymbol{V}(t+dt)]$ by $\Psi_{\mathbf{v}}^{(t+dt)}$ for all states $\mathbf{v} \in \hat{\mathcal{S}}$, we have to replace the condition $\{\boldsymbol{V}(t') = \mathbf{v}, t' \in [t, t+dt)\}$ by $\{\boldsymbol{V}(t+dt) = \mathbf{v}\}$ in the conditional expectation $\Phi_{\mathbf{v}}^{(t+dt)}$ that was computed in step 2. This is done by "distributing" $\Phi_{\mathbf{v}}^{(t+dt)}$ according to the change in the distribution of $\boldsymbol{V}(t)$ as explained below. The idea is to take into account that $\boldsymbol{V}(t)$ enters state $\mathbf{v}$ from $\mathbf{v}'$ during the interval $[t, t+dt)$. Assume that $[t, t+dt)$ is an infinitesimal time interval and that $q(\mathbf{v}', \mathbf{v}, dt)$, $\mathbf{v} \neq \mathbf{v}'$ is the probability to enter $\mathbf{v}$ from $\mathbf{v}'$ within $[t, t+dt)$. Then

$$\begin{aligned} Pr(\boldsymbol{V}(t+dt) = \mathbf{v}) = &\sum_{\mathbf{v}' \neq \mathbf{v}} q(\mathbf{v}', \mathbf{v}, dt) \cdot Pr(\boldsymbol{V}(t) = \mathbf{v}') \\ &+ \left(1 - \sum_{\mathbf{v}' \neq \mathbf{v}} q(\mathbf{v}, \mathbf{v}', dt)\right) \cdot Pr(\boldsymbol{V}(t) = \mathbf{v}) \,. \end{aligned} \tag{8.5}$$

Thus, we approximate $E[\mathbf{W}(t+dt) \,|\, \boldsymbol{V}(t+dt) = \mathbf{v}]$ as

$$\begin{aligned} &\sum_{\mathbf{v}' \neq \mathbf{v}} \Phi_{\mathbf{v}'}^{(t+dt)} \cdot q(\mathbf{v}', \mathbf{v}, dt) \cdot Pr(\boldsymbol{V}(t) = \mathbf{v}' | \boldsymbol{V}(t+dt) = \mathbf{v}) \\ &\quad + \Phi_{\mathbf{v}}^{(t)} \cdot (1 - \sum_{\mathbf{v}' \neq \mathbf{v}} q(\mathbf{v}, \mathbf{v}', dt)) \cdot Pr(\boldsymbol{V}(t) = \mathbf{v} | \boldsymbol{V}(t+dt) = \mathbf{v}) \,. \end{aligned} \tag{8.6}$$

Obviously, we can make use of the current approximations $\hat{\boldsymbol{p}}^{(t)}$ and $\hat{\boldsymbol{p}}^{(t+dt)}$ to compute the conditional probabilities $Pr(\boldsymbol{V}(t) = \mathbf{v}' | \boldsymbol{V}(t+dt) = \mathbf{v})$. For a small time step dt,

$$q(\mathbf{v}', \mathbf{v}, dt) \approx dt \cdot \alpha_j(\mathbf{v}', \Psi_{\mathbf{v}'}^{(t)})$$

if $\mathbf{v}' = \mathbf{v} - \hat{\mathbf{d}}_j$ and $q(\mathbf{v}', \mathbf{v}, dt) \approx 0$ otherwise.

Using Equation (8.6), we compute the approximation $\Psi_{\mathbf{v}}^{(t+dt)} \approx E[\mathbf{W}(t+dt) | \boldsymbol{V}(t+dt) = \mathbf{v}]$ as

$$\begin{aligned} \Psi_{\mathbf{v}}^{(t+dt)} = &\sum_{j=1}^{m} \Phi_{\mathbf{v}-\hat{\mathbf{d}}_j}^{(t+dt)} \cdot \frac{\hat{p}_{\mathbf{v}-\hat{\mathbf{d}}_j}^{(t)}}{\hat{p}_{\mathbf{v}}^{(t+dt)}} \cdot \alpha_j(\mathbf{v} - \hat{\mathbf{d}}_j, \Psi_{\mathbf{v}-\hat{\mathbf{d}}_j}^{(t)}) \cdot dt \\ &+ \Phi_{\mathbf{v}}^{(t+dt)} \cdot \frac{\hat{p}_{\mathbf{v}}^{(t)}}{\hat{p}_{\mathbf{v}}^{(t+dt)}} \left(1 - \sum_{j=1}^{m} \alpha_j(\mathbf{v}, \Psi_{\mathbf{v}}^{(t)})\right) \cdot dt). \end{aligned} \tag{8.7}$$

Note that the sum runs over all direct predecessors $\mathbf{v} - \hat{\mathbf{d}}_j$ of $\mathbf{v}$.

Example 8.1. *We consider a gene regulatory network, called the exclusive switch [75]. It consists of two genes with a common promotor region. Each of the two gene products P_1 and P_2 inhibits the expression of*

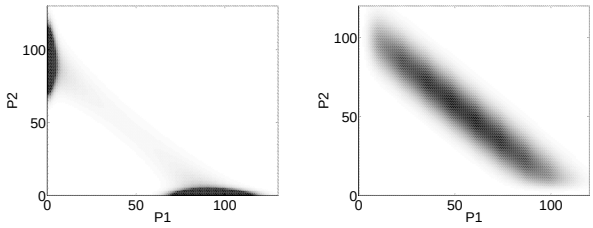


Figure 8.1: Probability distribution of the exclusive switch in Example 1 for two different parameter combinations.

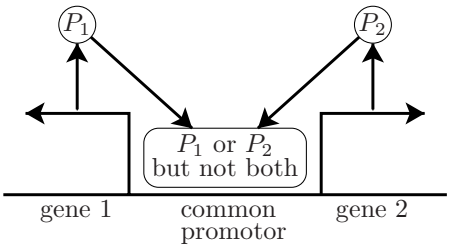


Figure 8.2: Illustration of the exclusive switch in Ex. 1 (picture is adapted from [75]). The stochastic hybrid model with only three discrete stochastic states and two differential equations per state.

the other product if a molecule is bound to the promotor region. More precisely, if the promotor region is free, molecules of both types P_1 and P_2 are produced. If a molecule of type P_1 (P_2) is bound to the promotor region, only molecules of type P_1 (P_2) are produced, respectively. We illustrate the network in Figure 11.2. The system has five chemical species of which two have an infinite range, namely P_1 and P_2. If $\mathbf{s} = (s_1, \ldots, s_5)$ is the current state, then the first two entries represent the populations of P_1 and P_2, respectively. The entry s_3 denotes the number of unbound DNA molecules which is either zero or one. The entry s_4 (s_5) is one if a molecule of type P_1 (P_2) is bound to the promotor region and zero otherwise. The chemical reactions are as follows. Let $j \in \{1, 2\}$.

- *We describe production of P_j by $DNA \to DNA + P_j$. Thus, $\mathbf{d}_j = \mathbf{e}_j - \mathbf{e}_3 + \mathbf{e}_3$ and $\alpha_j(\mathbf{s}) = c_j \cdot s_3$.*
- *We describe degradation of P_j by $P_j \to \emptyset$ with $\mathbf{d}_{j+2} = -\mathbf{e}_j$ and $\alpha_{j+2}(\mathbf{s}) = c_{j+2} \cdot s_j$.*
- *We model the binding of P_j to the promotor by $DNA + P_j \to DNA.P_j$ with $\mathbf{d}_{j+4} = -\mathbf{e}_j - \mathbf{e}_3 + \mathbf{e}_{j+3}$ and $\alpha_{j+4}(\mathbf{s}) = c_{j+4} \cdot s_j \cdot s_3$.*
- *For unbinding of P_j we use $DNA.P_j \to DNA + P_j$ with $\mathbf{d}_{j+6} = \mathbf{e}_j + \mathbf{e}_3 - \mathbf{e}_{j+3}$ and $\alpha_{j+6}(\mathbf{s}) = c_{j+6} \cdot s_{j+3}$.*

– *Finally, we have production of P_j if a molecule of type P_j is bound to the promotor, i.e., $DNA.P_j \to DNA.P_j + P_j$ with $\mathbf{d}_{j+8} = \mathbf{e}_j - \mathbf{e}_{j+3} + \mathbf{e}_{j+3}$ and $\alpha_{j+8}(\mathbf{s}) = c_{j+8} \cdot s_{j+3}$.*

Depending on the chosen parameters, the probability distribution of the exclusive switch is bistable, i.e. most of the probability mass concentrates on two distinct regions in the state space. In particular, if binding to the promotor is likely, then these two regions correspond to the two configurations where either the production of P_1 or the production of P_2 is inhibited. We illustrate the dynamics of the exclusive switch in Figure 8.1 by plotting the probability distribution for two different parameter combinations.

The expected number of molecules of type P_1 and/or P_2 may become high, depending on the chosen parameters. If, for instance, $c_1 = c_2 = c_9 = c_{10} = 0.5$, $c_3 = c_4 = c_7 = c_8 = 0.005$, $c_5 = c_6 = 0.01$, and we start initially without any proteins, i.e. with probability one in state $\mathbf{y} = (0, 0, 1, 0, 0)$, then after 500 time units most of the probability mass is located around the states $\mathbf{s} = (92, 2, 0, 1, 0)$ and $\mathbf{s} = (2, 92, 0, 0, 1)$ (compare the plot in Figure 8.1, left). Note that $s_3 = 0, s_4 = 1, s_5 = 0$ refers to the case that a molecule of type P_1 is bound to the promotor and $s_3 = s_4 = 0, s_5 = 1$ refers to the case that a molecule of type P_2 is bound to the promotor. Since for these parameters the system is symmetric, the expected populations of P_1 and P_2 are identical. Assume that at a certain time instant, both populations reach the threshold from which on we approximate them by continuous deterministic variables (we consider the unsymmetric case later, in Section 8.4). The remaining discrete stochastic model then becomes finite since only P_1 and P_2 have an infinite range in the original model ($\bar{n} = 2, \hat{n} = 3$). More precisely, it contains only 3 states, namely the state $\mathbf{v}_1$ where the promotor is free ($s_3 = 1, s_4 = s_5 = 0$), the state $\mathbf{v}_2$ where P_1 is bound to the promotor ($s_3 = 0, s_4 = 1, s_5 = 0$), and the state $\mathbf{v}_3$ where P_2 is bound to the promotor ($s_3 = s_4 = 0, s_5 = 1$), see also Figure 11.2. The differential equations which are used to approximate the conditional expectations $\Psi_{\mathbf{v}_1}^{(t+dt)}$, $\Psi_{\mathbf{v}_2}^{(t+dt)}$, and $\Psi_{\mathbf{v}_3}^{(t+dt)}$ are

$$\begin{aligned}
\frac{d\phi_{\mathbf{v}_1,j}^{(t')}}{dt'} &= \quad c_j - c_{2+j} \cdot \phi_{\mathbf{v}_1,j}^{(t')} - c_{4+j} \cdot \phi_{\mathbf{v}_1,j}^{(t')} \\
\frac{d\phi_{\mathbf{v}_2,j}^{(t')}}{dt'} &= -c_{2+j} \cdot \phi_{\mathbf{v}_2,j}^{(t')} + (c_7 + c_9) \cdot (2 - j) \\
\frac{d\phi_{\mathbf{v}_3,j}^{(t/)}}{dt'} &= -c_{2+j} \cdot \phi_{\mathbf{v}_3,j}^{(t')} + (c_8 + c_{10}) \cdot (j - 1)
\end{aligned}$$

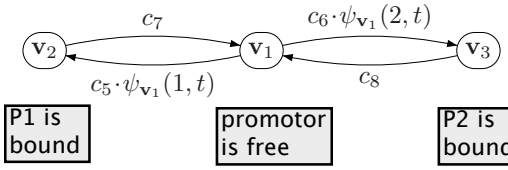


Figure 8.3: The discrete stochastic part of the stochastic hybrid model of Example 1.

where $\phi^{(t')}_{\mathbf{v},1}$ and $\phi^{(t')}_{\mathbf{v},2}$ are the elements of the vector $\Phi^{(t')}_{\mathbf{v}}$ representing the populations of P_1 and P_2, respectively ($\mathbf{v} \in \{\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3\}$). Since $j \in \{1, 2\}$ each of the three states has a system of two differential equations, one for P_1 and one for P_2. The transition rates in the discrete stochastic part of the model are illustrated in Figure 8.3. Thus, after solving the differential equations above to compute $\Phi^{(t+dt)}_{\mathbf{v}}$ for each $\mathbf{v}$ we obtain the vector $\Psi^{(t+dt)}_{\mathbf{v}}$ of the two conditional expectations for P_1 and P_2 from distributing $\Phi^{(t+dt)}_{\mathbf{v}_1}$, $\Phi^{(t+dt)}_{\mathbf{v}_2}$, $\Phi^{(t+dt)}_{\mathbf{v}_3}$ among the three states as defined in Equation (8.7)*. For the parameters used in Figure 11.2, left, the conditional expectations of the states $\mathbf{v}_2$ and $\mathbf{v}_3$ accurately predict the two stable regions where most of the probability mass is located. The state $\mathbf{v}_1$ has small probability and its conditional expectation is located between the two stable regions. It is important to point out that, for this example, a purely deterministic solution cannot detect the bistability because the deterministic model has a single steady-state [75]. Finally, we remark that in this example the number of states in the reduced discrete model is very small. If, however, populations with an infinite range but small expectations are present, we use the threshold abstraction described in Section 3.4 to keep the number of states small.*

If at time t a population, say the i-th population, is represented by its conditional expectations, it is possible to switch back to the original discrete stochastic treatment. This is done by adding an entry to the states $\mathbf{v}$ for the i-th dimension. This entry then equals $\psi^{(t)}_{\mathbf{v},i}$. This means that at this point we assume that the conditional probability distribution has mass one for the value $\psi^{(t)}_{\mathbf{v},i}$. Note that here switching back and forth between discrete stochastic and continuous deterministic representations is based on a population threshold. Thus, if the expectation of a population oscillates we may switch back and forth in each period.

We summarize the basic steps of our hybrid method for a time step of length dt in Algorithm 8.1. We assume that the sets $\bar{D}$ and $\hat{D}$ at time t are given and as in the previous chapter, $\mathbf{S}$ is a data structure that has nodes x for each states $x.\mathbf{v}$ that have significant probability, the field $x.p$ contains the probability of the state, and $x.\Psi$, $x.\Phi$ contain the conditional expectation of the state. The steps

Algorithm 8.1 A single iteration step of the hybrid algorithm.

```
1: choose step size dt;
2: update all x.p from t to t + dt by integrating Equation (8.3);
3: for all x ∈ S do
4:    x.Φ ← x.Ψ;  // Set initial condition
5:    update x.Φ by integrating Equation (8.4)
6: for all x ∈ S do
7:    update x.Ψ according to Equation (8.7);
8: for all i ∈ D̂ do
9:    E_i ← Σ_{x∈S} x.v · x.p;
10:   if E_i ≥ b_i then
11:      D̂ = D̂ \ {i};
12:      D̄ = D̄ ∪ {i};
13:      S' ← ∅;
14:      for all x ∈ S do
15:         x' ← find(x̂.v, S)
16:         if x' = null then
17:            x'.v ← (x̂.v);
18:            x'.p ← x.p;
19:            x'.Ψ ← respective conditional expectation;
20:            S' ← S' ∪ {x'};
21:         else
22:            x'.p ← x'.px.p;
23:      S = S';
24: for all i ∈ D̄ do
25:    E_i ← Σ_{x∈S} x.Ψ · x.p;
26:    if E_i < b_i then
27:       D̄ ← D̄ \ {i};
28:       D̂ ← D̂ ∪ {i};
29:       for all x ∈ S do
30:          extend x.v by x.ψ_i;
31:          remove x.ψ_i in x.Ψ;
```

in Algorithm 8.1 are then used to compute $\hat{\boldsymbol{p}}^{(t+dt)}$, $\Psi_{\mathbf{v}}^{(t+dt)}$ and to update $\bar{D}$, $\hat{D}$, and $\mathbf{S}$. In our implementation, we choose

$$dt = \min_{x \in \mathbf{S}} \frac{1}{\sum_{j=1}^{m} \alpha_j(x.\mathbf{v}, x.\Psi)}$$

since this is the minimal average residence time of all states in $\mathbf{S}$. Moreover, we solved Equation (8.3) using the RK4 method described in Section 4.4. Note that in line 10 we test whether the i-th species should be represented as a deterministic variable. This is done by comparing the expectation with the population threshold b_i. If the index i becomes deterministic, we have to reduce the set $\mathbf{S}$ by removing the i-th entry of each node $x \in \mathbf{S}$. Let $\hat{\mathbf{v}}$ be the state

vector that results from $\mathbf{v}$ if only entries with indices in $\hat{D} \setminus \{i\}$ are considered. We compute the probability $\hat{p}_{\hat{\mathbf{v}}}^{(t+dt)}$ of $\hat{\mathbf{v}}$ at time $t + dt$ as

$$\hat{p}_{\hat{\mathbf{v}}}^{(t+dt)} = \sum_{\mathbf{v}: v_k = \hat{v}_k, k \neq i} \hat{p}_{\mathbf{v}}^{(t+dt)}$$

where the sum ranges over all vectors $\mathbf{v}$ that are equal to $\hat{v}$ except for the i-th entry. In lines 14-23 we execute this by updating the field p of each node of the new state space $\mathbf{S}'$. In line 19 we use the "unreduced" distribution to compute the expectation of the i-th species at time $t + dt$ under the condition that the current state is $\hat{\mathbf{v}}$, for all $\hat{\mathbf{v}}$ of the reduced state space. Clearly, if we change the representation of several species, the reduction of state space can be performed in a single loop. For simplicity, we omit this improvement in our pseudocode. In a similar way as we reduce the state space we expand it in lines 26-31 because the expectation of the i-th species is less than b_i. Recall that $\psi_{\mathbf{v},i}^{(t)}$ is the entry of the vector $\Psi_{\mathbf{v}}^{(t)}$ that represents the conditional expectation of the i-th species. We approximate the expectations of all species in $\bar{D}$ as

$$E[\mathbf{W}(t + dt)] \approx \sum_{x \in \mathbf{S}} x.\Psi \cdot x.p.$$

In line 30 we modify the marginal probability distribution of the random variable that represents the i-th species in such a way that, with probability $x.p$ it has the conditional expectation $x.\psi_i$.

8.3 Propagation Model

We now show how to express Equations (8.4) and (8.7) using a discrete-time propagation process $(\tilde{\boldsymbol{f}}^{(k)})_{(k \in \mathbb{N}_0)}$ of a propagation model $\tilde{N}$. For this propagation model $\tilde{N}$ we can then relate the valuation of $\tilde{\boldsymbol{f}}^{(k)}$ to the conditional aggregated solution of TCM M at time t by making the time transformation $t = h \cdot k$, where h is an integration step. For the simplicity of the presentation we assume that all integration steps are equal.

We define $\tilde{\Psi}_{\mathbf{v}}^{(t)} = \Psi_{\mathbf{v}}^{(t)} \cdot \hat{p}_{\mathbf{v}}^{(t)}$ and then, from equation(8.7) we have that:

pset	purely stochastic			stochastic hybrid						purely determ.	
	ex. time	$\|\mathbf{S}\|$	error	pop. thres.	ex. time	$\|\mathbf{S}\|$	m1	m2	m3	ex. time	m1
1a	11h 46min	$8 \cdot 10^5$	$7 \cdot 10^{-5}$	50	15sec	$4 \cdot 10^2$	0.005	0.2	0.30	1sec	0.03
				100	1min 50sec	$3 \cdot 10^3$	0.004	0.2	0.30		
1b	7min 43sec	$5 \cdot 10^4$	$7 \cdot 10^{-7}$	50	1min 19sec	$6 \cdot 10^3$	0.01	0.19	0.30	1sec	0.03
				100	2min 50sec	$3 \cdot 10^4$	0.01	0.19	0.30		
2	4h 51min	$2 \cdot 10^5$	$4 \cdot 10^{-5}$	50	25sec	$4 \cdot 10^2$	0.06	0.08	0.09	1sec	0.45
				100	28sec	$6 \cdot 10^2$	0.06	0.07	0.09		
3	2min 21sec	$7 \cdot 10^5$	$6 \cdot 10^{-5}$	50	18sec	$6 \cdot 10^3$	0.02	0.08	0.16	1sec	0.05
				100	1min 41sec	$4 \cdot 10^4$	0.01	0.05	0.12		

Table 8.1: Results for the exclusive switch example.

$$\tilde{\Psi}_{\mathbf{v}}^{(t+dt)} = \sum_{j=1}^{m} \Phi_{\mathbf{v}-\hat{\mathbf{d}}_j}^{(t+dt)} \cdot \hat{p}_{\mathbf{v}-\hat{\mathbf{d}}_j}^{(t)} \cdot \alpha_j(\mathbf{v} - \hat{\mathbf{d}}_j, \Psi_{\mathbf{v}-\hat{\mathbf{d}}_j}^{(t)}) \cdot dt$$
$$+\Phi_{\mathbf{v}}^{(t+dt)} \cdot \hat{p}_{\mathbf{v}}^{(t)}(1 - \sum_{j=1}^{m} \alpha_j(\mathbf{v}, \Psi_{\mathbf{v}}^{(t)}) \cdot dt). \tag{8.8}$$

From Equation (8.4) we have that

$$\Phi_{\mathbf{v}}^{(t+dt)} \cdot \hat{p}_{\mathbf{v}}^{(t)} = \tilde{\Psi}_{\mathbf{v}}^{(t)} + \sum_{j=1}^{m} \bar{\mathbf{d}}_j \cdot \alpha_j(\mathbf{v}, \Psi_{\mathbf{v}}^{(t)}) \cdot \hat{p}_{\mathbf{v}}^{(t)} \cdot dt.$$

and therefore, we can rewrite Equation (8.8) as:

$$\tilde{\Psi}_{\mathbf{v}}^{(t+dt)} = \sum_{j=1}^{m} \Phi_{\mathbf{v}-\hat{\mathbf{d}}_j}^{(t+dt)} \cdot \alpha_j(\mathbf{v}, \Psi_{\mathbf{v}-\hat{\mathbf{d}}_j}^{(t)}) \cdot \hat{p}_{\mathbf{v}-\hat{\mathbf{d}}_j}^{(t)} \cdot dt$$
$$+\tilde{\Psi}_{\mathbf{v}}^{(t)} + \sum_{j=1}^{m} \bar{\mathbf{d}}_j \cdot \alpha_j(\mathbf{v}, \Psi_{\mathbf{v}}^{(t)}) \cdot \hat{p}_{\mathbf{v}}^{(t)} \cdot dt$$
$$-\Phi_{\mathbf{v}}^{(t+dt)} \cdot \sum_{j=1}^{m} \alpha_j(\mathbf{v}, \Psi_{\mathbf{v}}^{(t)}) \cdot \hat{p}_{\mathbf{v}}^{(t)} \cdot dt. \tag{8.9}$$

For a TCM $M = \langle \mathcal{S}, \mathbf{y}, \{\mathcal{C}_1, \ldots \mathcal{C}_m\} \rangle$, and a species split $\left\{\hat{D}, \bar{D}\right\}$, and integration step h, we construct the propagation model $\tilde{N} = \left\langle \hat{\mathcal{S}}, \mathcal{M}, \zeta, \pi \right\rangle$ with:

– mass space $\mathcal{M} = [0,1] \times \mathbb{R}_{\geq 0}^{\bar{n}} \times \mathbb{R}_{\geq 0}^{\bar{n}}$, which allows the propagation of probabilities and of the functions $\tilde{\Psi}$ and Φ,

– initial mass vector $\zeta_{\mathbf{s}} = \begin{cases} \langle 1, \mathbf{s}_{\bar{D}}, \mathbf{s}_{\bar{D}} \rangle & \text{if } \mathbf{s} = \mathbf{y}, \\ \langle 0, 0 \rangle & \text{if } \mathbf{s} \neq \mathbf{y}, \end{cases}$

– edge function π is defined by its respective components $\pi^p, \pi^\Phi, \pi^{\tilde{\Psi}}$, where:

  – the probability edge function, in accordance with Equation (8.3):

$$\pi^p_{\mathbf{v} \to \mathbf{v}+\hat{\mathbf{d}}_j}(\langle p, \Phi, \tilde{\Psi} \rangle) = p \cdot \alpha_j \left(\mathbf{v}, \frac{\tilde{\Psi}}{p} \right) \cdot h$$

– the Φ-edge function, in accordance with Equation (8.4):

$$\pi^{\Phi}_{\mathbf{v}\to\mathbf{v}}(\langle p, \Phi, \tilde{\Psi}\rangle) = \sum_{j=1}^{m} \bar{d}_j \cdot \alpha_j(\mathbf{v}, \Phi) \cdot h$$

– the $\tilde{\Psi}$-edge function, in accordance with Equation (8.9):

$$\pi^{\tilde{\Psi}}_{\mathbf{v}\to\mathbf{v}+\hat{\mathbf{d}}_j}(\langle p, \Phi, \tilde{\Psi}\rangle) = \Phi \cdot \alpha_j\left(\mathbf{v}, \frac{\tilde{\Psi}}{p}\right) \cdot p \cdot h$$
$$\pi^{\tilde{\Psi}}_{\mathbf{v}\to\mathbf{v}}(\langle p, \Phi, \tilde{\Psi}\rangle) = \sum_{j=1}^{m} \bar{d}_j \cdot \alpha_j\left(\mathbf{v}, \frac{\tilde{\Psi}}{p}\right) \cdot h.$$

Just to demonstrate the expressiveness of PMs, we give the description of a second propagation model that approximates the same conditional aggregated solution, but in a different way. $\tilde{N}' = \left\langle \hat{\mathcal{S}}, \mathcal{M}, \zeta, \pi \right\rangle$ with:

– mass space $\mathcal{M} = [0,1] \times \mathbb{R}^{\bar{n}}_{\geq 0}$, which allows the propagation of probabilities and of the un-normalized conditional expectations $\tilde{\mu}$,

– initial mass vector $\zeta_{\mathbf{s}} = \begin{cases} \langle 1, \mathbf{s}_{\bar{D}}\rangle & \text{if } \mathbf{s} = \mathbf{y}, \\ \langle 0, 0\rangle & \text{if } \mathbf{s} \neq \mathbf{y}, \end{cases}$

– edge function π is defined by its respective components π^{p}, π^{μ}, where:

– the probability edge function, similar to Equation (8.3)

$$\pi^{p}_{\mathbf{v}\to\mathbf{v}+\hat{\mathbf{d}}_j}(\langle p, \tilde{\mu}\rangle) = p \cdot \alpha_j\left(\mathbf{v}, \frac{\tilde{\mu}}{p}\right)$$

– the $\tilde{\mu}$-edge function directly propagation the value of the change vector:

$$\pi^{\tilde{\mu}}_{\mathbf{v}\to\mathbf{v}+\hat{\mathbf{d}}_j}(\langle p, \tilde{\mu}\rangle) = \left(\frac{\tilde{\mu}}{p} + \mathbf{d}_j\right) \cdot p \cdot \alpha_j\left(\mathbf{v}, \frac{\tilde{\mu}}{p}\right)$$
$$\pi^{\tilde{\mu}}_{\mathbf{v}\to\mathbf{v}}(\langle p, \tilde{\mu}\rangle) = -\sum_{j=1}^{m} \mathbf{d}_j \cdot p \cdot \alpha_j\left(\mathbf{v}, \frac{\tilde{\mu}}{p}\right).$$

And here, it is the continuous-time behaviour that matches the original TCM, i.e. $\mathbf{g}'^{(t)} \approx p^{(t)}$.

8.4 Experimental Results

We implemented the numerical solution of the stochastic hybrid model described above in C++ as well as the fast RK4 solution of the discrete stochastic model described in Section 4.4. In our implementation we dynamically switch the representation of a random variable whenever it reaches a certain population threshold. We ran experiments with two different thresholds (50 and 100) on an Intel 2.5GHz Linux workstation with 8GB of RAM. In this section we present 3 examples to that we applied our algorithm, namely the exclusive switch (see also Example 8.1), Goutsias' model, and a predator-prey model. Our most complex example has 6 different chemical species and 10 reactions. We compare our results to a purely stochastic solution where switching is turned off as well as to a purely deterministic solution. For all experiments, we fixed the cutting threshold $\delta = 10^{-14}$ to truncate the infinite state space as explained in Section 3.4.

Exclusive Switch. We chose different parameters for the exclusive switch in order to test whether our hybrid approach works well if

1) the populations of P_1 and P_2 are large (a) or small (b),
2) the model is unsymmetric (e.g. P_1 is produced at a higher rate than P_2 and degrades at a slower rate than P_2),
3) the bistable form of the distribution is destroyed (i.e. promotor binding is less likely, unbinding is more likely).

The following table lists the parameter sets (psets):

pset	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}
$1a$	5	5	0.0005	0.0005	0.1	0.1	0.005	0.005	5	5
$1b$	0.5	0.5	0.0005	0.0005	0.1	0.1	0.005	0.005	0.5	0.5
2	5	0.5	0.0005	0.005	0.1	0.1	0.005	0.005	5	0.5
3	0.5	0.5	0.0005	0.0005	0.01	0.01	0.1	0.1	0.5	0.5

We chose a time horizon of $t = 500$ for all parameter sets. Note that in the case of pset 3 the probability distribution forms a thick line in the state space (compare the plot in Figure 8.1, right). We list our results in Table 8.1 where the first column refers to the parameter set. Column 2 to 4 list the results of a purely stochastic solution where "ex. time" refers to the execution time, $|\mathbf{S}|$ to the average size of the set of significant states and "error" refers to the amount of probability mass lost due to the truncation with threshold δ, i.e. $1 - \sum_{x \in \mathbf{S}} x.p.$ The columns 6-10 list the results of our stochastic hybrid approach and column 5 lists the population threshold used for switching the representations in the stochastic hybrid model. Here, "m1", "m2", "m3" refer to the relative error of

model	purely stochastic			stochastic hybrid					purely determ.	
	ex. time	$\|\mathbf{S}\|$	error	pop. thres.	ex. time	$\|\mathbf{S}\|$	m1	m2	m3	ex. time	m1
Goutsias	1h 16min	$1 \cdot 10^6$	$4 \cdot 10^{-7}$	50	8min 47sec	$1 \cdot 10^5$	0.001	0.07	0.13	1sec	0.95
				100	48min 57sec	$6 \cdot 10^5$	0.0001	0.0003	0.001		
p.-prey	6h 6min	$5 \cdot 10^5$	$1 \cdot 10^{-7}$	50	8min 56sec	$2 \cdot 10^4$	0.06	0.15	0.27	1sec	0.86
				100	1h 2min	$8 \cdot 10^4$	0.04	0.11	0.23		

Table 8.2: Results for Goutsias' model and the predator-prey model.

the first three moments of the joint probability distribution at the final time instant. For this, we compare the (approximate) solution of the hybrid model with the solution of the purely stochastic model. Since we have five species, we simply take the average relative error over all species. Note that even if a species is represented by its conditional expectations, we can approximate its i-th moment by

$$E[\mathbf{W}(t)^i] \approx \sum_{x \in \mathbf{S}} (x.\Psi)^i \cdot x.p$$

where the i-th power of the vectors are taken component-wise. Finally, in the last two columns we list the results of a purely deterministic solution according to Equation (6.9). The last column refers to the average relative error of the expected populations when we compare the purely deterministic solution to the purely stochastic solution. Note that the deterministic solution of the exclusive switch yields an accurate approximation of the first moment (except for pset 2) because of the symmetry of the model. It does, however, not reveal the bistability of the distribution. As opposed to that, the hybrid solution *does* show this important property. For pset 1 and 3, the conditional expectations of the 3 discrete states are such that two of them match exactly the two stable regions where most of the probability mass is located. The remaining conditional expectation of the state where the promotor region is free has small probability and predicts a conditional expectation between the two stable regions. The execution time of the purely stochastic approach is high in the case of pset 1a, because the expected populations of P_1 and P_2 are high. This yields large sizes of $\mathbf{S}$ while we iterate over time. During the hybrid solution, we switch when the populations reach the threshold and the size of $\mathbf{S}$ drops to 3. Thus, the average number of significant states is much smaller. In the case of pset 1b, the expected populations are small and we use a deterministic representation for protein populations only during a short time interval (at the end of the time horizon). For pset 2, the accuracy of the purely deterministic solution is poor because the model is no longer symmetric. The accuracy of the hybrid solution on the other hand is less dependent on the symmetry of the model. Finally, for pset 3 the purely stochastic solution is fast because the production and binding rates are smaller compared to the other psets and fewer reactions occur per time unit. For a population threshold of 100 the hybrid model rarely uses a deterministic representation of the protein populations. Therefore the speed-up is small in that case.

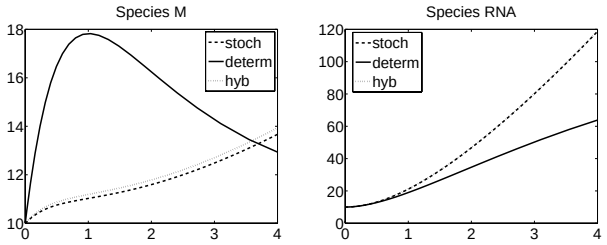


Figure 8.4: Expected populations in Goutsias' model.

Goutsias' Model. We reconsider Example 4.2 with parameters that differ from the original parameters used in [44] in that they increases the number of RNA molecules (because with the original parameters, all populations remain small).

c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}
0.043	7e-4	71.5	3.9e-6	0.02	0.48	2e-4	9e-12	0.08	0.5

Table 8.2 shows the results for Goutsias' model where we use the same column labels as above. We always start initially with 10 molecules of RNA, M, and D, as well as 2 DNA molecules. We choose the time horizon as $t = 4$. Note that the hybrid solution as well as the purely deterministic solution are feasible for much longer time horizons. The increase of the size of the set of significant states makes the purely stochastic solution infeasible for longer time horizons. As opposed to that the memory requirements of the hybrid solution remain tractable. In Figure 8.4 we plot the means of two of the six species obtained from the purely stochastic (stoch), purely deterministic (determ), and the hybrid (hyb) solution. Note that a purely deterministic solution yields very poor accuracy (average relative error of the means is about 95%).

Predator Prey. We apply our algorithm to the predator prey model described in [38]. It involves two species A and B and the reactions are $A \to 2A$, $A + B \to 2B$, and $B \to \emptyset$. The model shows sustainable periodic oscillations until eventually one of the populations reaches zero. We use this example to test the switching mechanism of our algorithm. We choose rate constants $c_1 = 1$, $c_2 = 0.03$, $c_3 = 1$ and start initially with 30 molecules of type A and 120 molecules of type B. For a population threshold of 50, we start with a stochastic representation of A and a deterministic representation of B. Then, around time 1.3 we switch to a purely stochastic representation since the expectation of B becomes less than 50. Around time $t = 6.1$ we switch the representation of A because $E[A(t)] > 50$, etc. We present our detailed results in Table 8.2. Similar to Goutsias' model, the deterministic solution has a high relative error whereas

the hybrid solution yields accurate results even though they are less accurate than the results for Goutsias' model. The reason is that the prey population becomes very large and its large variance is not adequately represented by the small number of discrete states. For instance, at the final time instant $t = 10$ the expected population size of prey is around 16,000.

Discussion. Our experimental results show that for the examples that we considered the hybrid approach is faster than the purely stochastic approach and more accurate than the purely deterministic approach. Clearly, more complex case studies have to be made to substantiate the usefulness of our approach in practice. Our hybrid solution will always be at last as accurate as a purely deterministic solution. An estimation of the approximation error, however, is difficult because currently no useful error estimates are known. The correlations between the random variables give hints about the linear dependencies but a direct relation to the approximation error has not yet been established. We believe, however, that for most examples, a simple population threshold is sufficient to obtain a solution that is much more accurate than a purely deterministic solution.

Chapter 9

Future Work: Tail Approximation

In this chapter we define the tail aggregated solution of a chemical master equation and then we propose a scheme for its approximation. We do not give the entire algorithm but we solve one of its fundamental components: estimating probabilities in the tail of a probability distribution.

9.1 Tail Aggregated Solution

We first define tail aggregation with respect to a tail boundary $\boldsymbol{b}$, and then, based on this aggregation we define the tail aggregated solution of a chemical master equation.

Definition 9.0.4 (Tail Aggregation)**.** *For a boundary $\boldsymbol{b} \in \mathbb{N}^n_{>0}$ we define the* tail aggregation $\hat{\mathcal{S}}_{\boldsymbol{b}}$ *to be:*

$$\hat{\mathcal{S}}_{\boldsymbol{b}} = \{0 \dots b_1 - 1, \top_{b_1}\} \times \{0 \dots b_2 - 1, \top_{b_2}\} \times \cdots \times \{0 \dots b_n - 1, \top_{b_n}\},$$

where the notation $\top_{b_i}$ denotes the interval $[b_i \dots \infty)$. Each element of $\hat{\mathcal{S}}_{\boldsymbol{b}}$ is called a tail aggregated state*.*

Through an abuse of notation, we let each vector in $\hat{\mathcal{S}}_{\boldsymbol{b}}$ represent a subset of $\mathcal{S}$. That is, for all $A \in \hat{\mathcal{S}}_{\boldsymbol{b}}$ we have that:

$$A = \{\mathbf{s} \in \mathcal{S} \mid A_i = s_i \text{ or } (s_i \geq b_i \text{ and } A_i = \top_{b_i})\},$$

from which it follows that the aggregation $\hat{\mathcal{S}}_{\boldsymbol{b}}$ defines a partition of $\mathcal{S}$.

Example 9.1. *For a two dimensional system, with* $\boldsymbol{b} = \{3, 3\}$*, the set tail aggregated partition of* $[0 \dots 6]^2$ *is shown in Figure 9.1. The element* $(1, \top_3)$ *represents the subset* $\{(1, s) \mid s \geq 3\}$*.*

Definition 9.0.5. *The* tail aggregated solution *of a CME with solution* $\boldsymbol{p}^{(t)}$*, with respect to the boundary* $\boldsymbol{b}$*, is a tuple* $\left\langle \hat{\mathbf{p}}^{(t)}, \boldsymbol{\mu}^{(t)} \right\rangle$*, such that:*

– $\hat{\mathbf{p}}^{(t)}$ *is an aggregated probability vector in* $\left[\hat{\mathcal{S}}_{\boldsymbol{b}} \to [0, 1]\right]$

$$\hat{p}_A^{(t)} = \sum_{\mathbf{s} \in A} p_{\mathbf{s}}^{(t)},$$

– *and* $\mu^{(t)}$ *is a conditional expectation vector in* $\left[\hat{\mathcal{S}}_{\boldsymbol{b}} \to \mathbb{R}^n\right]$

$$\boldsymbol{\mu}_A^{(t)} = \sum_{\mathbf{s} \in A} p_{\mathbf{s}}^{(t)} \cdot \mathbf{s}.$$

In order to obtain the exact tail aggregated solution of a chemical master equation, we must first compute the probabilities $p_{\mathbf{s}}^{(t)}$ and then aggregate them, according to definition 9.0.5. However, if we are only interested in an approximation of the aggregated solution, a direct method is possible, and usually desirable due to the state space reduction from $\mathcal{S}$ to $\hat{\mathcal{S}}_{\boldsymbol{b}}$, which leads to a large saving in the computation time.

9.2 Algorithmic Scheme

First, we formulate an aggregated chemical master equation for any partition $\hat{\mathcal{S}}$ of the state space $\mathcal{S}$. Then, we show how this aggregated equation could be solved numerically when the partition of the state space is the tail partition as defined above.

We now define $\tilde{\mathcal{S}}$ to be the coarsest refinement of $\hat{\mathcal{S}}$ for which, for each reaction R_j, the R_j successors of the states belonging to a partition set of $\tilde{\mathcal{S}}$ also belong to the same partition set of $\tilde{\mathcal{S}}$. Formally:

Definition 9.0.6. *Let* $\hat{\mathcal{S}}$ *be a partition of the state space* $\mathcal{S}$*, and, for all* $\mathbf{s} \in \mathcal{S}$*, let* $\hat{\mathbf{s}}$ *denote the element of* $\hat{\mathcal{S}}$ *for which* $\mathbf{s} \in \hat{\mathbf{s}}$*. For a transition class model* M *defined over the state space* $\mathcal{S}$*, we define an equivalence relation* $\simeq$ *over the states in* $\mathcal{S}$*.*

$$\mathbf{s} \simeq \mathbf{s}' \text{ iff } \hat{\mathbf{s}} = \hat{\mathbf{s}'} \text{ and } \widehat{u_j(\mathbf{s})} = \widehat{u_j(\mathbf{s}')}, \forall j = 1 \dots m.$$

We define $\tilde{\mathcal{S}}$ to be the set of equivalence classes induced on the space $\mathcal{S}$ by the equivalence relation $\simeq$, and we let $\tilde{\mathbf{s}} \in \tilde{\mathcal{S}}$ denote the equivalence class that contains $\mathbf{s} \in \mathcal{S}$.

As any two states in a $\simeq$-equivalence class $\tilde{\mathbf{s}}$ have successors in the same $\simeq$-equivalence class we can define the update function of a $\simeq$-equivalence class:

$$\tilde{u}_j(\tilde{\mathbf{s}}) = \widetilde{u_j(\mathbf{s})}.$$

In order to compute the aggregated solution of a transition class model, we need to solve the equation:

$$\frac{dp_A^{(t)}}{dt} = \sum_{j=1}^{m} \sum_{\mathbf{s}':u_j(\mathbf{s}')\in A} p_{\mathbf{s}'}^{(t)} \cdot \alpha_j(\mathbf{s}') - \sum_{j=1}^{m} \sum_{\mathbf{s}:\mathbf{s}\in A} p_{\mathbf{s}}^{(t)} \cdot \alpha_j(\mathbf{s}), \tag{9.1}$$

If, for each $j = 1 \dots m$, there exists a function $\tilde{\alpha}_j : \mathbb{R}^n_{\geq 0} \to \mathbb{R}$ such that

$$\sum_{k=1}^{\infty} c_k \cdot \alpha_j(s_k) = \tilde{\alpha}_j(\sum_{k=1}^{\infty} c_k \cdot s_k),$$

then, Equation (9.1) becomes:

$$\begin{aligned}\frac{dp_A^{(t)}}{dt} = &\sum_{j=1}^{m} \sum_{\tilde{\mathbf{s}}':\tilde{u_j}(\tilde{\mathbf{s}}')\subseteq A} p_{\tilde{\mathbf{s}}'}^{(t)} \cdot \tilde{\alpha}_j(\mu_{\tilde{\mathbf{s}}'}^{(t)}) \\ &- \sum_{j=1}^{m} \sum_{\tilde{\mathbf{s}}:\tilde{\mathbf{s}}\subseteq A} p_{\tilde{\mathbf{s}}}^{(t)} \cdot \tilde{\alpha}_j(\mu_{\tilde{\mathbf{s}}}^{(t)}).\end{aligned} \tag{9.2}$$

Let us consider $\hat{\mathcal{S}}_b$, the tail partition of the state space $\mathcal{S}$ of a transition class model M with respect to the boundary $\boldsymbol{b} \in \mathbb{N}^n_{>0}$. And let $\tilde{\mathcal{S}}_b$ be the refinement of $\hat{\mathcal{S}}_b$ according to Definition 9.0.6.

We split the states in the state space $\mathcal{S}$ into three classes:

$$\begin{aligned}\textbf{white states:} \quad & \mathcal{W} = \{\mathbf{s} \in \mathcal{S} \,|\, \hat{\mathbf{s}} = \{\mathbf{s}\}\}, \\ \textbf{gray states:} \quad & \mathcal{G} = \{\mathbf{s} \in \mathcal{S} \,|\, \tilde{\mathbf{s}} = \{\mathbf{s}\} \text{ and } \mathbf{s} \notin \mathcal{W}\}, \\ \textbf{black states:} \quad & \mathcal{B} = \mathcal{S} \setminus (\mathcal{W} \cup \mathcal{G}).\end{aligned}$$

In Section 9.3 we give a way to approximate the probabilities of gray states, probabilities that are needed by an algorithm that would implement Equation (9.2).

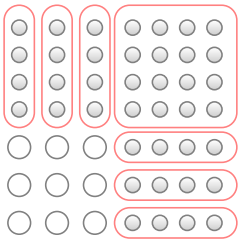


Figure 9.1: Partitioning of the state space used for an aggregated solution.

9.3 Tail Approximation

The key observation behind the tail approximation is that almost all probability distributions that describe the stochastic behaviour of real life have a certain "continuity" property and that their tail matches the shape of a geometric distribution. We do not formalize these properties here, but we refer to such real life distributions as "well-behaved", in order to distinguish them from other, possibly random, distributions.

Problem 9.1 (Tail approximation)**.** *For an aggregated solution* $\langle \hat{\boldsymbol{p}}, \boldsymbol{\mu} \rangle$ *with boundary* $b \in N$*, of an unknown probability distribution* p*, approximate the probabilities* $p_{\mathbf{s}}$ *of the states* $\mathbf{s}$ *that belong to the gray set* $\mathcal{G}$*.*

We first solve the tail approximation problem for systems of dimension one.

The tail approximation is done in two stages. First, we make a coarse approximation using a shifted geometric distribution and then we iteratively correct this value with respect to the probability values $\hat{p}_s$ with $s < b$.

Recall that the geometric distribution with mean M is a discrete probability distribution defined as:

$$\gamma_M(s) = p \cdot (1-p)^s, \text{ where } p = \frac{1}{M+1}.$$

For the first stage of our approximation, we define the *shifted geometric distribution* $\bar{\gamma}_{M,b} : \mathbb{N}_{\geq b} \to [0,1]$ to be: $\bar{\gamma}_{M,b}(s) = \gamma_{M-b}(s-b)$. This function is used to roughly approximate p_b by $\tilde{p}_b = \hat{p}(\top) \cdot \bar{\gamma}_{\mu,b}(b)$, where $\hat{p}(\top)$ is the probability for $x > b$, and $\Gamma_{\mu,b}$ approximates the probability of being in state b knowing conditioned on $x \geq b$ and knowing that the expectation for $x > b$ is μ.

The correction stage of our approximation is based on the observation that for a well-behaved probability distribution the relative errors of our approximation at points b and $b-1$ are almost equal:

$$\frac{\tilde{p}_b}{p_b} \approx \frac{\tilde{p}_{(b-1)}}{\hat{p}_{(b-1)}}.$$

And therefore, from the above approximation, we obtain the first corrected approximation $\tilde{p}^{c_1}$:

$$p_b \approx \tilde{p}_b^{c_1} = \tilde{p}_b \cdot \frac{\hat{p}_{(b-1)}}{\tilde{p}_{(b-1)}}.$$

This new value of the approximation can be updated in a second correction step in which the value $\hat{p}_{(b-2)}$ is taken into account. In Section 9.3.2 we present results for up to three correction steps.

Let us extend the vector $\hat{p}$ with $\hat{p}_{-1} = 0$, and let $z < b$ be the largest value for which $p_z = 0$. The correction stage of our approximation can have up to $b - z - 1$ steps.

9.3.1 Multiple dimensions

Here we are interested in the approximation of the value $p_{\mathbf{s}}$ with $\mathbf{s} \in \mathbb{N}_0^n$. First, we define the one dimensional sub-stochastic vector $p_{\mathbf{s}\,|\,i} : \mathbb{N}_0 \to [0,1]$. Let $\mathbf{s}_{\,|\,i}$ be the set of states that are equal to $\mathbf{s}$ on all components except i, and let $\mathbf{s}_{\,|\,i,x}$ be the state $\mathbf{s}$ where the i-th component is changed to value x. Now, $p_{\mathbf{s}\,|\,i}$ is a vector with positive values over $\mathbf{s}_{\,|\,i}$ with entries $p_{\mathbf{s}\,|\,i,x} = p_{\mathbf{s}\,|\,i,x}$.

For a state $\mathbf{s}$ with only one component i for which $s_i = \top$ we apply the 1-dimension tail approximation method on the projection $\langle \hat{p}_{\mathbf{s}\,|\,i}, \mu_i \rangle$.[1]

Finally, if we have more than one dimensions for which $s_i = \top$, we make an independence assumption in order to reduce the problem to one dimension. For 2-dimensions, this assumption is:

$$p^{(t)}_{(b,b)} \approx \hat{p}^{(t)}_{(b,\top)} \cdot \hat{p}^{(t)}_{(\top,b)}.$$

9.3.2 Case Studies

In this section we give statistical evidence that show the level of accuracy of our approximation. For this, we start with the actual solution of the CME of two different biochemical reaction networks, from which we compute the aggregated

[1] The tail approximation can be applied on sub-stochastic vectors as well.

Table 9.1: Results.

Model	b	Max abs. err.			rel. err.			abs. err.$> 1\times10^{-7}$		
		1st corr.	2nd corr.	3rd corr.	1st corr.	2nd corr.	3rd corr.	1st corr.	2nd corr.	3rd corr.
pred. prey	10	2×10^{-3}	1×10^{-3}	5×10^{-4}	8×10^{-2}	3×10^{-2}	1×10^{-2}	46%	46%	43%
	50	2×10^{-4}	2×10^{-5}	1×10^{-6}	2×10^{-1}	2×10^{-2}	9×10^{-4}	4%	4%	3%
	100	4×10^{-8}	1×10^{-9}	6×10^{-11}	1×10^{-3}	6×10^{-5}	2×10^{-6}	0%	0%	0%
ex. swtich	10	7×10^{-3}	1×10^{-2}	1×10^{-3}	4×10^{-1}	4×10^{-1}	1×10^{-1}	92%	95%	94%
	50	1×10^{-3}	1×10^{-4}	2×10^{-5}	1×10^{-1}	1×10^{-2}	1×10^{-3}	46%	46%	12%
	100	3×10^{-4}	3×10^{-5}	3×10^{-6}	4×10^{-2}	4×10^{-3}	4×10^{-4}	33%	33%	14%

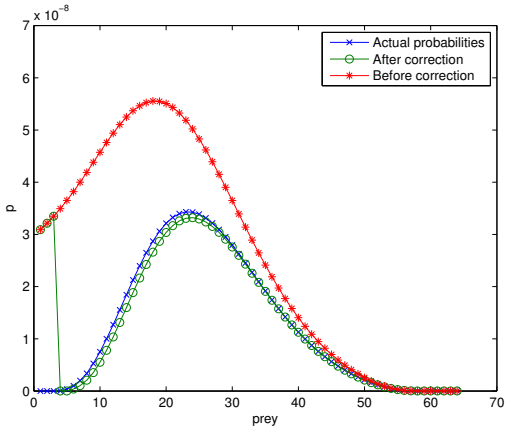


Figure 9.2: For all boundaries $b \in [1 \dots 64]$, we show the comparison between the actual probability vector p_b, the first approximation $\tilde{p}_b$ and the corrected approximation $\tilde{p}_b^{c_1}$. The probability vector p is taken from the solution at time $t = 3$ of a predator-prey system and it gives the probabilities over the number of preys for a fixed number of 46 predators.

Figure 9.3: Zoom on the same comparison as in Figure 9.2.

solution with respect to a boundary b (step in which we loose information). After that, we use the tail approximation in order to restore the probabilities of the gray states $\mathcal{G}$ from the aggregated solution. Finally, we compare the restored probabilities with those in the probability distribution we have started with.

We consider two systems: the predator-prey[38] and the exclusive switch[75]. First, we compute the solution of the CME associated to each of these two systems (at time points $t = 1 \dots 5$ for predator-prey and $t = 10, 20 \dots 100$ for the exclusive switch) using our previous algorithm[23] and tool[24]. For each of the obtained probability distributions $\boldsymbol{p}^{(t)}$, for each of the boundaries $b \in \{10, 50, 100\}$, and for all states $\mathbf{s}$ in the boundary set $\mathcal{G}$, we consider all projections $p_{\mathbf{s}\,|\,i}$ of the solution $\boldsymbol{p}^{(t)}$. It is these projections that we first aggregate and then restore using tail approximation.

For three correction steps $k = 1, 2, 3$, the column **max. abs. err.** of table 9.1 gives the maximal absolute error computed as

$$\max_{\mathbf{s} \in \mathcal{G}} \Big(\,|\, p_{\mathbf{s}}^{c_k} - p_{\mathbf{s}} \,|\, \Big).$$

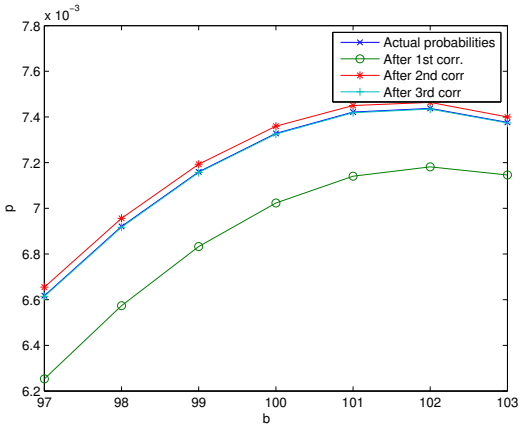


Figure 9.4: Approximation of state probabilities in the exclusive switch model. With each correction step the approximation is closer to the real probabilities.

The column **rel. err.** gives the relative error for the state with maximal absolute error. The relative error is computed as:

$$1 - \min\left(\frac{\tilde{p}_{\mathbf{s}}^{c_k}}{p_{\mathbf{s}}}, \frac{p_{\mathbf{s}}}{\tilde{p}_{\mathbf{s}}^{c_k}}\right).$$

Finally, table 9.1 also reports the percentage of projections for which the absolute error, as defined above, is greater than $1 \boxtimes 10^{-7}$. In some cases this percentage is high because the boundary is too small.

In Figure 9.2 we show how the tail approximation performs at all possible boundaries b of a predator-prey model for the projection $p_{\,|\,\text{predator}=46}$. For this figure, the original probability distribution has been aggregated with respect to a different boundary b for each possible value of preys. For small boundaries, the region of the probability distribution at the right of the boundary does not have a geometric shape and thus the errors are larger. Even more, for very small boundaries, the correction step can not be applied because the value of p is 0, and the approximation is completely unacceptable. This is one of the major problems that an algorithm using this approximation needs to solve. However, for larger boundaries, the approximation is very accurate and the correction step is performing well.

Figure 9.4 illustrates the case in which more than one correction steps are needed in order to obtain and accurate result. As future work, we hope to design a fix-point algorithm that would detect how many correction steps are necessary for a given tolerance.

Chapter 10

Related Work

Different hybrid approaches have been proposed in the literature [50, 90, 98]. As opposed to our approach, they focus on Monte Carlo simulation and consider the problem of multiple time scales. They do not use deterministic variables but try to reduce the computational complexity of generating a trajectory of the model by approximating the number of reactions during a certain time step.

The closest work to ours is the hybrid approach proposed by Hellander and Lötstedt [54]. They approximate large populations by normally distributed random variables with a small variance and use Monte Carlo simulation to statistically estimate the probability distribution of the remaining populations with small sizes. They consider a single ODE to approximate the expected sizes of the large populations. As opposed to that, here we consider a set of ODEs to approximate the expected sizes of the large populations *conditioned* on the small populations. This allows us to track the dependencies between the different populations more acurately. Moreover, instead of a statistical estimation of probabilities, we provide a direct numerical method to solve the stochastic hybrid model. The direct numerical method that we use for the computation of the probability distributions of the stochastic variables has shown to be superior to Monte Carlo simulation [22]. Another difference is that the method in [54] does not allow a dynamic switching between stochastic and deterministic treatment of variables.

Finally, our approaches are related to the stochastic hybrid models considered in [11, 15] and to fluid stochastic Petri nets [63]. These approaches differ from our approach in that they use probability distributions for the different values a continuous variable can take. In our setting, at a fixed point in time we only consider the conditional expectations of the continuous variables, which

is based on the assumption that the respective populations are large and their relative variance is small. This allows us to provide an efficient numerical approximation algorithm that can be applied to systems with large state spaces. The stochastic hybrid models in [11, 15, 63] cannot be solved numerically except in the case of small state spaces.

Chapter 11

SABRE

11.1 Introduction

In this chapter we present SABRE, a tool for stochastic analysis of biochemical reaction networks. SABRE implements fast adaptive uniformization (FAU), a direct numerical approximation algorithm for computing transient solutions of biochemical reaction networks. Biochemical reactions networks represent biological systems studied at a molecular level and these reactions can be modeled as transitions of a Markov chain. SABRE accepts as input the formalism of guarded commands, which it interprets either as continuous-time or as discrete-time Markov chains. Besides operating in a stochastic mode, SABRE may also perform a deterministic analysis by directly computing a mean-field approximation of the system under study. SABRE does not currently offer an implementation of the hybrid methods introduced in the second part of this thesis. We plan to add these methods to our tool in a future release.

Numerical analysis tools for discrete-state Markov processes such as PRISM[72], INFAMY[47], ETMCC[59], MRMC[67], APNNtoolbox[10], SHARPE[60], SPNP[61], or Möbius[18] have been introduced (see Section 11.6). However, except for INFAMY, these tools do not accept models with possibly infinite state space. It is important to note that many population models have an infinite state space, that is, the number of reachable states is infinite. Even when in the real system the number of molecules, or more generally, individuals is finite, no a priori bound is known, and models do not include any constraints on the number of molecules, for example in production rules such as $\emptyset \to A$. Another issue is that existing tools usually implement algorithms that are not optimized specifically for population models, and do not scale well on such models.

SABRE is a tool for the transient analysis of Markov population models. In other words, SABRE analysis discrete-time, or continuous-time Markov processes that have a structured discrete state space and state-depended rate functions. In Section 11.2 we give more details on the space structure and the state dependency of rate functions that are present in Markov processes that represent population models.

SABRE offers both stochastic and deterministic analysis of population models. For stochastic analysis, SABRE implements three algorithms: standard uniformization, fast adaptive uniformization and Runge-Kutta fourth order method. The different configurations in which SABRE may operate are depicted in Figure 11.1. The focus of the tool is on the fast adaptive uniformization method, while the remaining methods are given for completeness and comparison.

SABRE is available on-line at http://mtc.epfl.ch/~mateescu/sabre. First, the user gives an input model (either in SBML format or in guarded commands format) and a time horizon and than the transient analysis of the system starts (see Figure 11.3). More details on the usage of the tool are given in Section 11.3.

11.2 Guarded Commands

Guarded-command models (GCM) is the input formalism of SABRE. GCMs are a textual description of processes and are given in the style of Dijkstra's guarded-command language[25]. Their syntax has subsequently been used by languages such as Reactive Modules [4] and by the language for specifying PRISM models[96]. The basic unit within GCMs is a transition class, which is expressed as a guarded command that operates on the state variables of the system. A transition class encodes for a possibly infinite number of state transitions. Within population models, the state variables of the system are non-negative integers representing numbers of molecules for each species. A guarded command takes the form

```
guard |- rate -> update
```

where the `guard` is a Boolean predicate over the variables that determines in which states the corresponding transitions are enabled. The `update` is a rule that describes the change of the system variables if the transition is performed. Syntactically, `update` is a list of statements, each assigning to a variable an expression over variables. Assume that `x` is a variable. If, for instance, the update rule is that `x` is incremented by 1, we write `x:=x+1`. We assume that

variables that are not listed in the update rule do not change if the transition is taken. Each guarded command also assigns a `rate` to the corresponding transitions, which is a function on the state variables. Within SABRE, `rate` is given in infix notation. In the case of population models, the update function is incrementing or decrementing each variable by a constant integer.

For a population model with m reactions, the GCM description is a set of m guarded commands, which we index as $\texttt{guard}_j \vdash \texttt{rate}_j \rightarrow \texttt{update}_j$, where each of the commands j, with $1 \leq j \leq m$, describe the j-th reaction of the model.

GCMs are used to express both CTMCs and DTMCs. The difference between the two interpretations comes from the semantic given to the rate function of each command. In the case of CTMCs, for a given reaction j, the rate function $\texttt{rate}_j$ assigns to each state $\mathbf{s}$, a positive real value that represents the rate of the outgoing transition j.

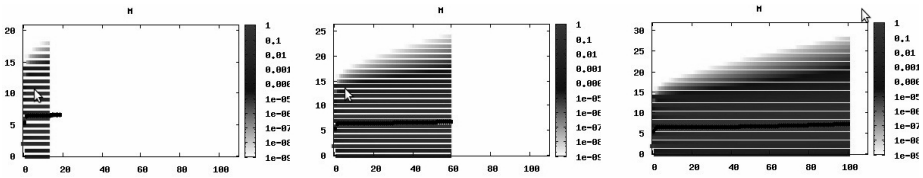
In the case of DTMCs, the rate function $\texttt{rate}_j$ assigns to each state $\mathbf{s}$, a positive real value that represents the transition probability from state $\mathbf{s}$ to its successor on reaction j. The functions $\texttt{rate}_j$ must define probability distribution over the direct successor state, that is, for each state $\mathbf{s}$ we impose that Σ_j $\texttt{rate}_j(\mathbf{s}) = 1$. If the input is not given in this manner, SABRE will automatically normalize the rate functions such that the probability distribution condition to be fulfilled. Note that this is equivalent to interpreting the input as a CTMC and than considering its embedded DTMC.

GCMs are used to model systems that exhibit a finite number of transition types, but possibly an unbounded number of states. For example, in a computer network, the number of type of events is finite (send message, receive message, add node, etc.) but the number of states is countably infinite, because it depends on the number of nodes in the network and on the number of requests each of them has. The same holds for biochemical reaction networks, each reaction type generates a transition class, but the number of states is countably infinite, as we do not have any a-priori bound on the variables of the system, due to productions rules of the type $\emptyset \rightarrow A$. We therefore conclude that GCMs are a natural formalism for describing population models[57]

Example 11.1. *The bistable toggle switch is a prototype of a genetic switch with two competing repressor proteins and four reactions. We call the species A and B and we let $x = (x_A, x_B) \in \mathbb{N}_0^2$ be a vector describing a state of the system. The reactions are given in Table 11.1.*

Table 11.1: Simple toggle switch example

Reaction	**Guarded command**		
$\emptyset \to A$	true	$\vdash c_1/(c_2 + x_B^2)$	$\to x_A := x_A + 1$
$A \to \emptyset$	$A > 0$	$\vdash c_3 \cdot x_1$	$\to x_A := x_A - 1$
$\emptyset \to B$	true	$\vdash c_4/(c_5 + x_A^2)$	$\to x_B := x_B + 1$
$B \to \emptyset$	$B > 0$	$\vdash c_6 \cdot x_2$	$\to x_B := x_B - 1$



11.3 Tool Interface

From the tool's interface, we have several ways of selecting a model for analysis. One can load an existing model, upload an SBML file or introduce a GCM text description of the system to analyze. SBML is a standardized format for representing models of biological processes, such as metabolism or cell signaling and is the input to SABRE's core program. GCMs that have update functions with constant increment (or decrement) have a straight forward translation to SBML.

Example 11.2. *We continue the toggle switch example with its SBML description. For brevity, we only give one reaction of the model. We observe that the rate function is not restricted to a particular template and is written following the mathML standard.*

```
 0 <sbml ...>
 1 <model>
 1 ...
 3 <listOfSpecies>
 4   <species id="A" initialAmount="133"/>
 5   <species id="B" initialAmount="133"/>
 6 </listOfSpecies>
 7 <listOfReactions>
 8   <reaction id="R1">
 9     <listOfProducts>
10       <speciesReference species="A"/>
11     </listOfProducts>
```

```
12     <listOfModifiers>
13       <speciesReference species="B"/>
14     </listOfModifiers>
15     <kineticLaw>
16        <math ...>
17          <apply> <divide/>
18            <ci> c1 </ci>
19            <apply> <plus/>
20              <ci> c2 </ci>
21              <apply> <times/>
22                <ci> B </ci>
23                <ci> B </ci>
24              </apply>
25            </apply>
26          </apply>
27        </math>
28        <listOfParameters>
29        <parameter id="c1" value="3000"/>
30        <parameter id="c2" value="11000"/>
31        </listOfParameters>
32     </kineticLaw>
33   </reaction>
34   ...
35 </listOfReactions>
36 </model>
36 </sbml>
```

Once the model is chosen, we choose a configuration of the analysis by choosing the semantics, the mode and, if needed, the type of stochastic solution. Finally, we choose a time horizon, or the number of steps for which we want the system to run. We also give as an input a dump time t_d, which corresponds to the intermediate results, that is, the system will compute the distributions for $t_d, 2 \cdot t_d, \cdots t$. The program computes the intermediates and the final results which are then dynamically plotted for each species, as the computation runs (see Figure 11.3). If the uniformization method is selected, the user also needs to provide an estimate of the maximal exit rate over all reachable states. If the estimate is too small, the compuation needs to be restarted, and if the estimate is too large, the computation is likely to take longer. It is standard

uniformization which is especially touched by choosing a too large upper bound on the maximal exit rate. Estimating this upper bound by heuristics such as those used for the sliding window algorithm[56] is an on going work.

11.4 Software Architecture

SABRE is available on line, assuring this was a fast and portable release of our implementation. The core of our tool is implemented in C++, while the website that hosts it is implemented using PHP and Javascript. The user provides the desired input through the web interface, than a query is generated to the 3GHz Linux machine on which SABRE is installed. The server sends back to the user intermidiate results which are then plotted as we show in Section 11.3.

11.4.1 Components

SABRE's different components are activated as shown in Figure 11.1. Depending on the chosen semantics, analysis mode and, if necessary, stochastic solution type, SABRE calls the coresponding method. Some of the functionalities are shared among different methods, for example the DTMC solution is accessed either directly from choosing the DTMC semantics, either indirectly, by the uniformization algorithm. As well, Runge-Kutta method, is used both as a solver of the CME or as the solver of the reaction rate equations.

11.4.2 Data Structure

We present an efficient data structure used by SABRE when used in stochastic analysis mode. SABRE's main focus is on a fast implementation of the fast adaptive algorithm, so we will use this algorithm to motivate the choice of our data structure. However, the same kind of reasoning works if one wants to optimize the Runge-Kutta implementation. The most computationally demanding part of fast adaptive uniformization is the probability propagation phase, which performs the equivalent of one matrix-vector product in a DTMC. We therefore need a data structure that is efficient during this step.

First, we mention that, for each state, along with the state description, we need to record additional information about the probability of the state, about its successors, and about the rates/probabilities of the reactions that lead to those respective successors. We gather all this information in a structure called `node`. During the propagate phase we iterate over all nodes of the state

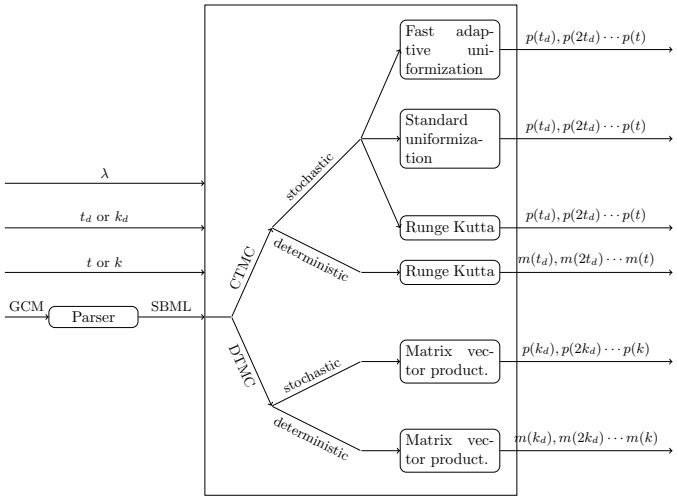


Figure 11.1: Software architecture. Depending on the selected semantics, analysis mode and, eventually, type of stochastic solution, SABRE computes the desired results. The vector $p(t)$ is the transient probability distribution after time t, while the vector $p(k)$ is the transient probability distribution after k steps. For the deterministic analysis, the values $m(t)$ and $m(k)$ correspond to the mean field of the corresponding CTMC, respectively DTMC. The value λ represents the maximum exit rate of the CTMC and is required only by uniformization.

space, and for each node we move probability mass along all of its outgoing transitions. Note that, initially the state space has a single state, and that states are dynamically added to the state space as they are discovered. That is, some of the direct successors of n may be newly discovered, and in this case they are added to the state space data structure. Therefore, ideally, the data structure used for storing information about the state space would have the following characteristics.

– [Fast sequential access.] For enumerating all nodes. We note that this is a property of the array primitive type of most programming languages.

– [Fast search.] For quickly finding the successors of a node. We note that this is a property of map or hash type of many programming languages.

– [Fast add.] For dynamically adding newly discovered states to the current state space.

– [Fast delete.] For dynamically removing states that have close to zero probability.

Table 11.2: Data structure comparison

Data Structure	**Sequential access**	**Search**	**Add**	**Delete**
Arrays	fast	slow	fast	slow
Maps	slow	fast	fast	fast
Hybrid solution	fast	fast	complex, but fast	complex, but fast

We summarize the comparison between arrays and hashes in Table 11.2. Arrays allow fast sequential access, fast add but slow search and delete operations. Hashes allow fast add, delete, search, but slow sequential access. We propose a hybrid solution that has the advantages of each data structure at the expense of extra memory usage.

Our hybrid data structure is composed of:

– array `nodes` that acts as a function from index → node

– hash `index` that acts as a function from state → index

– vector `inactive_nodes` of indices of nodes that have become inactive as a result of a delete.

This mixture of structures lets us give fast implementations for each of the required operations:

– **Sequential access** Simple iteration over the elements of `nodes`.

– **Search** Search within `index` followed by an access in `nodes`.

– **Delete state** The `nodes` array is allocated statically, so physically erasing a node would be expensive. The alternative is to mark the node for deletion by inactivating it –setting its probability to zero– and adding it to the `inactive_nodes` vector. Because of their zero probability, inactive nodes are ignored when iterating over all states. An inactive node has two possible futures: either it will be reoccupied by a newly added state, either it will be deleted during a compress phase. The compress phase is initiated when the number of inactive nodes covers more then 20% of the number of both active nodes and inactive nodes and it consists of eliminating all inactive nodes and rearranging the active nodes in a contiguous region.

– **Add state** When we add a state to the state space, we need to assign it to a node within the `nodes` array. The `nodes` array is allocated statically and during the program's initialization phase, it is initialized to 2^{20} free nodes. When we add a new state, if `inactive_nodes` is non-empty, that is, if an

Table 11.3: Case Studies Summary

Analysis	Model	Time	Error	States
Stochastic	Exclusive switch	$94s$	$9e-8$	3047
Deterministic	Enzymatic reaction	$< 1s$	–	1
Stochastic	Moran's model	$49s$	0	1001

inactive node exist, assign the state to this node, which now becomes active. If `inactive_nodes` is empty, we check whether we still have free allocated nodes, that is, we check whether the number of active nodes has reached the size allocated to `nodes`. If free nodes exist, we assign the new state to a free node, if free nodes do not exist we need to allocate extra 2^{20} nodes to `nodes` and then pick a newly created free node. We note that the reallocation operation is expensive but happens only rarely, e.g. when the state space first reaches one million, two millions, three millions states and so on.

11.5 Case Studies

We present case studies for stochastic and deterministic analysis of CTMCs and for the stochastic analysis of DTMCs. For more and larger experiments on stochastic analysis of CTMCs we refer the reader to the paper giving the fast adaptive uniformization algorithm. All our experiments are performed on a 3GHz Intel Linux PC, with 6 GB of RAM. We give the results of our experiments in Table 11.3.

11.5.1 Genetic exclusive switch

The exclusive genetic switch we analyze involves two species of proteins that may bound to the same promoter site. We denote the unbounded proteins by N_1 and N_2 and the bounded ones by r_1 and r_2[8]. The guarded commands for this model are given in Table 11.4. The rate functions are evaluated for the state $(x_{N_1}, x_{r_1}, x_{N_2}, x_{r_2})$, where x_{N_1} is the number of molecules of type N_1 and so on.

When it is bounded to the promotor site, a protein represses the production of the other protein. And so, for example, production of N_1 only happens if no N_2 molecule is bounded to the promoter site (see rate function of first reaction). N_1 or N_2 may bound only to a free promotor site (see rate functions of the third and seventh reaction). Note that it always holds that $x_{r_1} + x_{r_2} \leq 1$.

Table 11.4: Genetic exclusive switch

Reaction	**Guarded Command**		
$\emptyset \rightarrow N_1$	true	$\vdash g_1 \cdot (1 - x_{r_2})$	$\rightarrow x_{N_1} := x_{N_1} + 1$
$N_1 \rightarrow \emptyset$	$x_{N_1} > 0$	$\vdash d_1 \cdot x_{N_1}$	$\rightarrow x_{N_1} := x_{N_1} - 1$
$N_1 \rightarrow r_1$	$x_{N_1} > 0$	$\vdash b_1 \cdot (1 - x_{r_1} - x_{r_2})$	$\rightarrow x_{N_1} := x_{N_1} - 1; x_{r_1} := x_{r_1} + 1$
$r_1 \rightarrow N_1$	$x_{r_1} > 0$	$\vdash u_1 \cdot x_{r_1}$	$\rightarrow x_{N_1} := x_{N_1} + 1; x_{r_1} := x_{r_1} - 1$
$\emptyset \rightarrow N_2$	true	$\vdash g_2 \cdot (1 - x_{r_1})$	$\rightarrow x_{N_2} := x_{N_2} + 1$
$N_2 \rightarrow \emptyset$	$x_{N_2} > 0$	$\vdash d_2 \cdot x_{N_2}$	$\rightarrow x_{N_2} := x_{N_2} - 1$
$N_2 \rightarrow r_2$	$x_{N_2} > 0$	$\vdash b_2 \cdot (1 - x_{r_1} - x_{r_2})$	$\rightarrow x_{N_2} := x_{N_2} - 1; x_{r_2} := x_{r_2} + 1$
$r_2 \rightarrow N_2$	$x_{r_2} > 0$	$\vdash u_2 \cdot x_{r_2}$	$\rightarrow x_{N_2} := x_{N_2} + 1; x_{r_2} := x_{r_2} - 1$

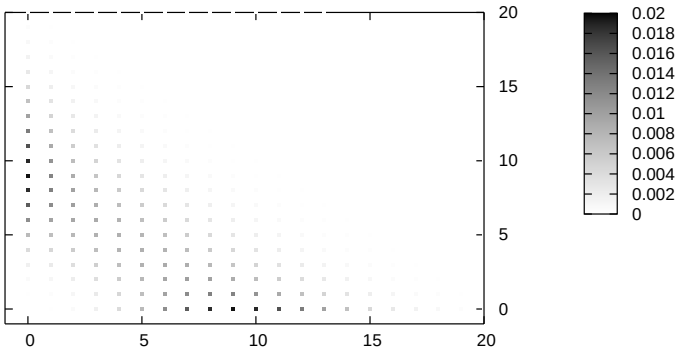


Figure 11.2: Exclusive switch at time 10000. The x-axis gives the number of N_1 molecules and the y-axis gives the number of N_2 molecules. Each point of the plot corresponds to the states of systems that have the have the corresponding number of N_1 and N_2 molecules. The darker the point is, the more probability mass it holds. We can notice the bistable behaviour from the two regions of black points, one for $N_1 = 0$ and one for $N_2 = 0$.

We run the system from initial state $(25, 0, 0, 0)$ for a period of time of 10000 units with constants: $g_1 = g_2 = 0.05, d_1 = d_2 = 0.005, b_1 = b_2 = 0.1, u_1 = u_2 = 0.005$, and present the solution in Figure 11.2

11.5.2 Enzymatic reaction

We use *enzyme-catalyzed substrate conversion* to exemplify how to perform a deterministic analysis under continuous-time semantics. The enzymatic reaction is described by three reactions (see Table 11.5), that involve four chemical species, namely, enzyme (E), substrate (S), complex (C), and product (P) molecules. The state of the system is described by the vector (x_E, x_S, x_C, x_P), which gives the existing number of molecules of each type.

For our experimental results, we chose the same parameters as in [12], that is, initial state $y = (1000, 100, 0, 0)$, time horizon $t = 70$, and rate constants

Table 11.5: Enzymatic reaction

Reaction	**Guarded Command**		
$E + S \to C$	$x_E > 0$ and $x_S > 0$	$\vdash c_1 \cdot x_E \cdot x_S$	$\to x_E := x_E - 1; x_S := x_S - 1; x_C := x_C + 1$
$C \to E + S$	$x_C > 0$	$\vdash c_2 \cdot x_C$	$\to x_E := x_E + 1; x_S := x_S + 1; x_C := x_C - 1$
$C \to E + P$	$x_C > 0$	$\vdash c_3 \cdot x_C$	$\to x_E := x_E + 1; x_C := x_C - 1; x_P := x_P + 1$

$c_1 = c_2 = 1$ and $c_3 = 0.1$. For the case deterministic analysis we can not give any error bounds, as shown in Table 11.3.

11.5.3 Moran's population model

As a simple example of how SABRE operates on DTMC models we choose Moran's genetic population model, which can be seen as a set of biochemical reactions, more specifically as one reversible reaction.

For a population of N individuals, with two alleles, A_1 and A_2, we are interested to find the probability of fixation of A_1, that is, the probability for A_1 individuals to be equal to N after a certain time. We have two reactions: $A_2 \to A_1$ and $A_1 \to A_2$. For x_{A_1} individuals with A_1 allele and x_{A_2} individuals with A_2 allele, the probability of the first reaction is $\frac{1-s}{2} + s \cdot \frac{x_{A_1}}{N}$, where s is a small constant. As for the second reaction, its probability is $\frac{1-s}{2} + s \cdot \frac{x_{A_2}}{N}$.

We choose $N = 1000$ and $s = 2e - 3$, the initial state of $x_{A_1} = 1$ and we perform a transient analysis until time $k = 10^6$, at this time, the probability of fixation is 0.00049. In this case the error we obtain is 0 because no cutting is performed, the state space is kept at its complete size of 1001.

11.6 Comparison with other tools

Several tools for stochastic analysis of Markov chains have been developed by communities such as probabilistic verification, computational biology and performance evaluation among others. Here, we provide a comparison with the tools that are the closest to SABRE. The PRISM tool [72], which is widely used in probabilistic verification, considers a more general class of Markov processes than population models. For instance, it does not restrict the update function such that it allows only a constant change of the state variables. The models addressed by PRISM are less structured and typically they do not have state dependent rate functions. PRISM uses powerful minimization techniques such as bisimulation that do not result in significant reductions in the case of population model. PRISM requires that upper bounds on the state variables are given as an

input by the user. As opposed to that the SABRE tool finds appropriate bounds automatically and avoids an exhaustive state space exploration. The drawback is that the SABRE tool cannot guarantee the validity of properties such as "Is the probability to reach state x within t time greater than p?" but gives an approximate solution. As opposed to that PRISM can guarantee such properties. On the other hand, since SABRE avoids an exhaustive state space exploration it is able to handle much larger models with state-dependent rates. Infamy is a model-checking tool for infinite-state CTMCs by Zhang et al. [47]. Depending on the desired precision, their algorithm simply explores the reachable states up to a finite path depth. In contrast, our approach takes into account the direction into which the probability mass moves, and constructs a sequence of abstract models "on-the-fly," during the verification process. Similar approaches have also been used in the context of biochemical reaction networks [12].

Other tools for stochastic analysis of Markov chains, such as ETMCC[59], MRMC[67], APNNtoolbox[10], SHARPE[60], SPNP[61], and Möbius[18], are conceived for answering performance analysis questions and as PRISM, due to their exhaustive state space exploration can not be applied to infinite models.

Dizzy[92], Snoopy[52] and Copasi[62] are tools for stochastic simulation alone and not do not compute probability distributions over states.[22] Bio-PEPA[17] is a language for modeling and analysis of biochemical networks. For numerical analysis and verification problems Bio-PEPA uses PRISM's engine.

Chapter 12

Conclusions

The careful reader has noticed that each new chapter was introduced on very similar lines with regards to what motivated that particular work. More specifically, each chapter introduced a new method to reduce the state space of our problem by keeping the accuracy of our solutions high, and each method would solve systems more efficiently than the previous one. Furthermore, most of our methods are applying techniques from the field of computer aided verification to biological probabilistic systems, such as abstract and on-the-fly techniques. Indeed, the numerical analysis for probabilistic systems can be seen as the counterpart of simulation, just as in software model checking verification is the counter part of testing. As opposed to non-deterministic model checking, probabilistic model-checking can make use of the information that some events occur with a very small probability in order to approximate the solution of the system. Reachability is the building block for software model checking, and similarly the transient analysis of Markov chains together with steady-state analysis (which we have not addressed here) are the building boxes for probabilistic model checking, where from the importance of these problems.

We have started with the sliding window method which solves continuous-time Markov chains by observing only the state space that fits under the current window of the algorithm. This window is dynamically adjusted and shifted in order to follow the significant probability mass of the system. Then, we have moved to methods that deal with the space explosion problem in discrete-time and not in continuous-time. This switch allows us to keep a very precise account of the significant support of our solution through threshold abstraction and on-the-fly techniques. Also, these techniques were presented in the general framework of propagation models.

In the second part of this thesis we have introduce hybrid methods, which are even more aggressive in reducing the state space of the system by directly reducing its dimensions. These methods are justified by the nature of biological systems where stochasticity is induced mainly by molecules that come in small numbers, and thus a stochastic representation is not needed for molecules that are available in large numbers with high probability.

Propagation models were introduce in order to offer a common data structure for our algorithms. This data structure is defined over mass vectors and offers a propagation operator that moves mass through the space of the system. We have seen that this mass can be probability mass, can represent expectation of variables, but it is not restricted to any of these type of values. We have classified propagation models in linear and non-linear classes and the results presented in this paper for non-linear propagation models are purely theoretical, as our implemented algorithms only use linear propagation models.

As future work, the propagation models framework can be enhanced in several different directions such as composition or different aggregation techniques. As well, due to their strong connection to differential equations, we believe that propagation models might find application outside of systems biology problems. We also leave as future work an algorithm that uses the tail approximation technique or other customized hybrid algorithms.

References

[1] P. Abdulla, C. Baier, S. Iyer, and B. Jonsson. Reasoning about probabilistic lossy channel systems. In *Proc. CONCUR'00*, LNCS, pages 320–333. Springer, 2000.

[2] P. Abdulla, N. Bertrand, A. Rabinovich, and P. Schnoebelen. Verification of probabilistic systems with faulty communication. *Inf. Comput.*, 202(2):141–165, 2005.

[3] P. Abdulla, N. B. Henda, and R. Mayr. Verifying infinite Markov chains with a finite attractor or the global coarseness property. In *Proc. LICS '05*, pages 127–136. IEEE Computer Society, 2005.

[4] Rajeev Alur and Thomas A. Henzinger. Reactive modules. *Formal Methods in System Design*, 15(1):7–48, 1999.

[5] A. Arkin, J. Ross, and H. H. McAdams. Stochastic kinetic analysis of developmental pathway bifurcation in phage λ-infected E. coli cells. *Genetics*, 149:1633–1648, 1998.

[6] G.A. Baker. *The essentials of Padé approximants*. Academic Press, New York, 1975.

[7] N. Barkai and S. Leibler. Biological rhythms: Circadian clocks limited by noise. *Nature*, 403:267–268, 2000.

[8] Baruch Barzel and Ofer Biham. Calculation of switching times in the genetic toggle switch and other bistable systems. *Phys. Rev. E*, 78(4):041919, Oct 2008.

[9] P. Bremaud. *Markov Chains*. Springer, 1998.

[10] Peter Buchholz, Joost-Pieter Katoen, Peter Kemper, and Carsten Tepper. Model-checking large structured markov chains. *J. Log. Algebr. Program.*, 56(1-2):69–97, 2003.

[11] Manuela L. Bujorianu and John Lygeros. Towars a general theory of stochastic hybrid systems. In *Stochastic Hybrid Systems*, volume 337 of *Lecture Notes in Control and Information Sciences*, pages 3–30. Springer, 2006.

[12] K. Burrage, M. Hegland, F. Macnamara, and B Sidje. A Krylov-based finite state projection algorithm for solving the chemical master equation arising in the discrete modelling of biological systems. In *Proceedings of the Markov 150th Anniversary Conference*, pages 21–38. Boson Books, 2006.

[13] H. Busch, W. Sandmann, and V. Wolf. A numerical aggregation algorithm for the enzyme-catalyzed substrate conversion. In *Proc, of CMSB*, volume 4210 of *LNCS*, pages 298–311. Springer, 2006.

[14] Y. Cao, D. Gillespie, and L. Petzold. Efficient step size selection for the tau-leaping simulation method. *J. Chem. Phys.*, 124(4), 2006.

[15] C.G. Cassandras and J. Lygeros. Stochastic hybrid systems: Research issues and areas. In *Stochastic Hybrid Systems, (C.G. Cassandras, and J. Lygeros, Ed.s)*, pages 1–14. Taylor and Francis, 2006.

[16] E. Çinlar. *Introduction to Stochastic Processes*. Prentice-Hall, 1975.

[17] Federica Ciocchetta and Jane Hillston. Bio-pepa: A framework for the modelling and analysis of biological systems. *Theor. Comput. Sci.*, 410(33-34):3065–3084, 2009.

[18] David Daly, Daniel D. Deavours, Jay M. Doyle, Patrick G. Webster, and William H. Sanders. Möbius: An extensible tool for performance and dependability modeling. In *Computer Performance Evaluation / TOOLS*, pages 332–336, 2000.

[19] P. D'Argenio, B. Jeannet, H. Jensen, and K. Larsen. Reachability analysis of probabilistic systems by successive refinements. In *Proc. PAPM-PROBMIV'01*, pages 39–56, 2001.

[20] L. de Alfaro and R. Pritam. Magnifying-lens abstraction for Markov decision processes. In *Proc. CAV*, volume 4590 of *LNCS*, pages 325–338. Springer, 2007.

[21] E. de Souza e Silva and R. Gail. Transient solutions for Markov chains. In *Computational Probability*, chapter 3, pages 43–79. Kluwer Academic Publishers, 2000.

[22] Frédéric Didier, Thomas A. Henzinger, Maria Mateescu, and Verena Wolf. Approximation of event probabilities in noisy cellular processes. In *Proc. of CMSB*, volume 5688 of *LNBI*, page 173, 2009.

[23] Frédéric Didier, Thomas A. Henzinger, Maria Mateescu, and Verena Wolf. Fast adaptive uniformization of the chemical master equation. *High Performance Computational Systems Biology, International Workshop on*, 0:118–127, 2009.

[24] Frédéric Didier, Thomas A. Henzinger, Maria Mateescu, and Verena Wolf. Sabre: A tool for stochastic analysis of biochemical reaction networks. In *QEST*, pages 193–194, 2010.

[25] Edsger W. Dijkstra. Guarded commands, nondeterminacy and formal derivation of programs. *Commun. ACM*, 18(8):453–457, 1975.

[26] J. Dunkel and H. Stahl. On the transient analysis of stiff markov chains. In *Proocedings of the 3rd IFIP Working Conference on Dependable Computing for Critical Applications*, 1992.

[27] M. B. Elowitz, M. J. Levine, E. D. Siggia, and P. S. Swain. Stochastic gene expression in a single cell. *Science*, 297:1183–1186, 2002.

[28] S. Engblom. Galerkin spectral method applied to the chemical master equation. *Comm. Comput. Phys.*, 5:871–896, 2009.

[29] J. Esparza and K. Etessami. Verifying probabilistic procedural programs. In *Proc. FSTTCS'04*, volume 3328 of *LNCS*, pages 16–31. Springer, 2005.

[30] J. Esparza, A. Kucera, and R. Mayr. Model checking probabilistic pushdown automata. In *Proc. LICS '04*, pages 12–21. IEEE Computer Society, 2004.

[31] K. Etessami and M. Yannakakis. Algorithmic verification of recursive probabilistic state machines. In *Proc. TACAS'05*, LNCS, pages 253–270. Springer, 2005.

[32] N. Fedoroff and W. Fontana. Small numbers of big molecules. *Science*, 297:1129–1131, 2002.

[33] William Feller. *An Introduction to Probability Theory and Its Applications, Volume 1*. Wiley, January 1968.

[34] Lars Ferm, Per Lötstedt, and Andreas Hellander. A hierarchy of approximations of the master equation scaled by a size parameter. *Journal of Scientific Computing*, 34:127 – 151, 2008.

[35] B. L. Fox and P. W. Glynn. Computing Poisson probabilities. *Communications of the ACM*, 31(4):440–445, 1988.

[36] E. Gallopoulos and Y. Saad. On the parallel solution of parabolic equations. In *In Proc. ACM SIGARCH-89*, pages 17–28. ACM press, 1989.

[37] T. Gardner, C. Cantor, and J. Collins. Construction of a genetic toggle switch in Escherichia coli. *Nature*, 403:339 – 342, 2000.

[38] D. T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *J. Phys. Chem.*, 81(25):2340–2361, 1977.

[39] D. T. Gillespie. A rigorous derivation of the chemical master equation. *Physica A*, 188:404–425, 1992.

[40] D. T. Gillespie. Approximate accelerated stochastic simulation of chemically reacting systems. *J. Chem. Phys.*, 115(4):1716–1732, 2001.

[41] Daniel T. Gillespie. Simulation methods in systems biology. In *Formal Methods for Computational Systems Biology*, pages 125–167, 2008.

[42] Byron Goldstein, James R. Faeder, and William S. Hlavacek. Mathematical and computational models of immune-receptor signalling. *Nat. Rev. Immunol.*, 4, 2004.

[43] D. Gonze, J. Halloy, and A. Goldbeter. Robustness of circadian rhythms with respect to molecular noise. *PNAS, USA*, 99(2):673–678, 2002.

[44] J. Goutsias. Quasiequilibrium approximation of fast reaction kinetics in stochastic biochemical systems. *J. Chem. Phys.*, 122(18):184102, 2005.

[45] W. K. Grassmann, editor. *Computational Probability*. Kluwer Academic Publishers, 2000.

[46] D. Gross and D. Miller. The randomization technique as a modeling tool and solution procedure for transient Markov processes. *Operations Research*, 32(2):926–944, 1984.

[47] Ernst Moritz Hahn, Holger Hermanns, Björn Wachter, and Lijun Zhang. Infamy: An infinite-state markov model checker. In *CAV*, pages 641–647, 2009.

[48] E. Hairer, S. Norsett, and G. Wanner. *Solving Ordinary Differential Equations I: Nonstiff Problems*. Springer, 2008.

[49] E. Hairer and G. Wanner. *Solving Ordinary Differential Equations II. Stiff and Differential-Algebraic Problems*. Springer, 2004.

[50] E. L. Haseltine and J. B. Rawlings. Approximate simulation of coupled fast and slow reactions for chemical kinetics. *J. Chem. Phys.*, 117:6959–6969, 2002.

[51] M. Hegland, C. Burden, L. Santoso, S. Macnamara, and H. Booth. A solver for the stochastic master equation applied to gene regulatory networks. *J. Comput. Appl. Math.*, 205:708–724, 2007.

[52] Monika Heiner, Ronny Richter, and Martin Schwarick. Snoopy - a tool to design and animate/simulate graph-based formalisms. In *In Proc. PNTAP 2008, associated to SIMUTools 2008. ACM digital library*, 2008.

[53] A. Hellander. Efficient computation of transient solutions of the chemical master equation based on uniformization and quasi-Monte carlo. *J. Chem. Phys.*, 128(15):154109, 2008.

[54] A. Hellander and P. Lötstedt. Hybrid method for the chemical master equation. *Journal of Computational Physics*, 227, 2007.

[55] D. A. Henderson, R. J. Boys, C. J. Proctor, and D. J. Wilkinson. Linking systems biology models to data: a stochastic kinetic model of p53 oscillations. In A. O'Hagan and M. West, editors, *Handbook of Applied Bayesian Analysis*. Oxford University Press, 2009.

[56] T. Henzinger, M. Mateescu, and V. Wolf. Sliding window abstraction for infinite Markov chains. In *Proc. CAV*, volume 5643 of *LNCS*, pages 337–352. Springer, 2009.

[57] Thomas Henzinger, Barbara Jobstmann, and Verena Wolf. Formalisms for specifying markovian population models. In Springer, editor, *LIX Colloquium Reachability Problems'09*, pages 3–23, 2009.

[58] Thomas A. Henzinger, Barbara Jobstmann, and Verena Wolf. Formalisms for specifying Markovian population models. In *Proc. LIX Colloquium Reachability Problems*, volume 5797 of *LNCS*. Springer, 2009.

[59] Holger Hermanns, Joost-Pieter Katoen, Joachim Meyer-Kayser, and Markus Siegle. A tool for model-checking markov chains. *STTT*, 4(2):153–172, 2003.

[60] Christophe Hirel, Robin A. Sahner, Xinyu Zang, and Kishor S. Trivedi. Reliability and performability modeling using sharpe 2000. In *Computer Performance Evaluation / TOOLS*, pages 345–349, 2000.

[61] Christophe Hirel, Bruno Tuffin, and Kishor S. Trivedi. Spnp: Stochastic petri nets. version 6.0. In *Computer Performance Evaluation / TOOLS*, pages 354–357, 2000.

[62] Stefan Hoops, Sven Sahle, Ralph Gauges, Christine Lee, Jürgen Pahle, Natalia Simus, Mudita Singhal, Liang Xu, Pedro Mendes, and Ursula Kummer. Copasi - a complex pathway simulator. *Bioinformatics*, 22(24):3067–3074, 2006.

[63] Graham Horton, Vidyadhar G. Kulkarni, David M. Nicol, and Kishor S. Trivedi. Fluid stochastic Petri nets: Theory, applications, and solution techniques. *European Journal of Operational Research*, 105(1):184–201, 1998.

[64] A. Jensen. Markoff chains as an aid in the study of Markoff processes. *Skandinavisk Aktuarietidskrift*, 36:87–91, 1953.

[65] N. G. van Kampen. *Stochastic Processes in Physics and Chemistry.* Elsevier, 3rd edition, 2007.

[66] J.-P. Katoen, D. Klink, M. Leucker, and V. Wolf. Three-valued abstraction for continuous-time Markov chains. In *Proc. CAV*, volume 4590 of *LNCS*, pages 316–329. Springer, 2007.

[67] Joost-Pieter Katoen, Maneesh Khattri, and Ivan S. Zapreev. A markov reward model checker. In *QEST*, pages 243–244, 2005.

[68] A. Kucera. Methods for quantitative analysis of probabilistic pushdown automata. *Electr. Notes Theor. Comput. Sci.*, 149(1):3–15, 2006.

[69] T. G. Kurtz. The relationship between stochastic and deterministic models for chemical reactions. *J. Chem. Phys.*, 57(7):2976 –2978, 1972.

[70] T.G. Kurtz. *Approximation of Population Processes.* Society for Industrial Mathematics, 1981.

[71] M. Kwiatkowska, G. Norman, and D. Parker. Game-based abstraction for Markov decision processes. In *QEST*, pages 157–166. IEEE CS Press, 2006.

[72] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. Prism 2.0: A tool for probabilistic model checking. In *QEST*, pages 322–323, 2004.

[73] A. Law and D. Kelton. *Simulation Modelling and Analysis*. McGraw-Hill Education, 2000.

[74] J. W. Little, D. P. Shepley, and D. W. Wert. Robustness of a gene regulatory circuit. *The EMBO Journal*, 18(15):4299—-4307, 1999.

[75] Adiel Loinger, Azi Lipshtat, Nathalie Q. Balaban, and Ofer Biham. Stochastic simulations of genetic switch systems. *Phys. Rev. E*, 75(2):021904, 2007.

[76] H. Maamar, A. Raj, and D. Dubnau. Noise in gene expression determines cell fate in *Bacillus subtilis*. *Science*, 317(5837):526 – 529, 2007.

[77] R. A. Marie, A. L. Reibman, and K. S. Trivedi. Transient analysis of acyclic Markov chains. *Perform. Eval.*, 7(3):175–194, 1987.

[78] H. H. McAdams and A. Arkin. Stochastic mechanisms in gene expression. *PNAS, USA*, 94:814–819, 1997.

[79] H. H. McAdams and A. Arkin. It's a noisy business! *Trends in Genetics*, 15(2):65–69, 1999.

[80] C. Moler and C. Van Loan. Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later. *SIAM Review*, 45(1):3–49, 2003.

[81] C. B. Moler and C. F. Van Loan. Nineteen dubious ways to compute the exponential of a matrix. *SIAM Review*, 20(4):801–836, 1978.

[82] B. Munsky. *The Finite State Projection Approach for the Solution of the Master Equation and its Applications to Stochastic Gene Regulatory Networks*. PhD thesis, University of California, Santa Barbara, 2008.

[83] B. Munsky and M. Khammash. The finite state projection algorithm for the solution of the chemical master equation. *J. Chem. Phys.*, 124:044144, 2006.

[84] B. Munsky and M. Khammash. A multiple time interval finite state projection algorithm for the solution to the chemical master equation. *J. Comp. Phys.*, 226:818–835, 2007.

[85] J. Norris. *Markov Chains*. Cambridge University Press, 1. edition, 1999.

[86] P. Patel, B. Arcangioli, S. Baker, A. Bensimon, and N. Rhind. DNA replication origins fire stochastically in fission yeast. *Mol. Biol. Cell*, 17:308–316, 2006.

[87] J. Paulsson. Summing up the noise in gene networks. *Nature*, 427(6973):415–418, 2004.

[88] S. Peles, B. Munsky, and M. Khammash. Reduction and solution of the chemical master equation using time scale separation and finite state projection. *J. Chem. Phys.*, 125:204104, 2006.

[89] B. Philippe and R. Sidje. Transient solutions of Markov processes by Krylov subspaces. In *Proc. of the 2nd International Workshop on the Numerical Solution of Markov Chains*, pages 95–119. Kluwer Academic Publishers, 1995.

[90] Jacek Puchalka and Andrzej M. Kierzek. Bridging the gap between stochastic and deterministic regimes in the kinetic simulations of the biochemical reaction networks. *Biophysical Journal*, 86(3):1357 – 1372, 2004.

[91] A. Rabinovich. Quantitative analysis of probabilistic lossy channel systems. *Inf. Comput.*, 204(5):713–740, 2006.

[92] Stephen Ramsey, David Orrell, and Hamid Bolouri. Dizzy: Stochastic simulation of large-scale genetic regulatory networks. *J. Bioinformatics and Computational Biology*, 3(2):415–436, 2005.

[93] C. Rao, D. Wolf, and A. Arkin. Control, exploitation and tolerance of intracellular noise. *Nature*, 420(6912):231–237, 2002.

[94] A. Reibman and K. Trivedi. Numerical transient analysis of Markov models. *Comput. Oper. Res.*, 15(1):19–36, 1988.

[95] Anne Remke. *Model checking structured infinite Markov chains*. PhD thesis, Enschede, June 2008.

[96] J. Rutten, M. Kwiatkowska, G. Norman, and D. Parker. *Mathematical Techniques for Analyzing Concurrent and Probabilistic Systems*, volume 23 of *CRM Monograph Series*. American Mathematical Society, 2004.

[97] Y. Saad. Analysis of some krylov subspace approximations to the matrix exponential operator. *SIAM J. Numer. Anal.*, 29(1):209–228, 1992.

[98] H. Salis and Y. Kaznessis. Accurate hybrid stochastic simulation of a system of coupled chemical or biochemical reactions. *J. Chem. Phys.*, 122, 2005.

[99] W. Sandmann and V. Wolf. A computational stochastic modeling formalism for biological networks. In *Enformatika Transactions on Engineering, Computing and Technology*, volume 14, pages 132–137, 2006.

[100] R. Sidje, K. Burrage, and S. MacNamara. Inexact uniformization method for computing transient distributions of Markov chains. *SIAM J. Sci. Comput.*, 29(6):2562–2580, 2007.

[101] R. Sidje and W. Stewart. A survey of methods for computing large sparse matrix exponentials arising in Markov chains. In *Markov Chains, Computational Statistics and Data Analysis 29*, pages 345–368, 1996.

[102] Roger B. Sidje. EXPOKIT: Software package for computing matrix exponentials. *ACM Transactions on Mathematical Software*, 24(1):130–156, 1998.

[103] P. Sjöberg. *Numerical Methods for Stochastic Modeling of Genes and Proteins*. PhD thesis, Uppsala University, Sweden, 2007.

[104] P. Sjöberg, P. Lötstedt, and J. Elf. Fokker-Planck approximation of the master equation in molecular biology. *Computing and Visualization in Science*, 12:37–50, 2009.

[105] W. J. Stewart. *Introduction to the Numerical Solution of Markov Chains*. Princeton University Press, 1995.

[106] C. Strelen. Approximate disaggregation-aggregation solutions for general queueing networks. In *Society for Computer Simulation*, pages 773–778, 1997.

[107] P. S. Swain, M. B. Elowitz, and E. D. Siggia. Intrinsic and extrinsic contributions to stochasticity in gene expression. *PNAS, USA*, 99(20):12795–12800, 2002.

[108] M. Thattai and A. van Oudenaarden. Intrinsic noise in gene regulatory networks. *PNAS, USA*, 98(15):8614–8619, July 2001.

[109] A. van Moorsel and W. Sanders. Adaptive uniformization. *ORSA Communications in Statistics: Stochastic Models*, 10(3):619–648, 1994.

[110] A. Warmflash and A. Dinner. Signatures of combinatorial regulation in intrinsic biological noise. *PNAS*, 105(45):17262–17267, 2008.

[111] J. Zhang, L. T. Watson, and Y. Cao. A modified uniformization method for the solution of the chemical master equation. Technical Report TR-07-31, Computer Science, Virginia Tech., 2007.

[112] L. Zhang, H. Hermanns, E. Moritz Hahn, and B. Wachter. Time-bounded model checking of infinite-state continuous-time Markov chains. In *ACSD*, 2008. China.

Curriculum Vitae

Maria Mateescu

Education

2006 – 2011	Ph.D. in Computer Science Ecole Polytechnique Fédérale de Lausanne (EPFL)
1999 – 2005	M.Sc. in Computer Science "Politehnica" University of Bucharest

Experience

2011	Internship at Google as a Software Engineer (Zurich, 3 months)
2006 – 2010	Teaching Assistant for "Theoretical Computer Science" and other subjects Ecole Polytechnique Fédérale de Lausanne (EPFL)
2004 – 2005	Software Engineer Freescale Semiconductors, Bucharest
2003 – 2005	Teaching Assistant "Politehnica" University of Bucharest

Publications

1. T.A. Henzinger, and M. Mateescu, "Propagation Models for Computing Biochemical Reaction Networks", in the *Proceedings of the 9th International Conference on Computational Methods in Systems Biology (CMSB 2011) - to appear.*
2. T.A. Henzinger, and M. Mateescu, "Tail Approximation for the Chemical Master Equation", in the *Proceedings of the 8th International Workshop on Computational Systems Biology (WCSB 2011).*

3. F. Didier, T.A. Henzinger, M. Mateescu, and V. Wolf, "SABRE: A Stochastic Analysis Tool for Biochemical Reaction Networks", in the *Proceedings of the 7th International Conference on Quantitative Evaluation of SysTems (QEST 2010)*, IEEE Computer Society, 2010.

4. T.A. Henzinger, L. Mikeev, M. Mateescu, and V. Wolf, "Hybrid Numerical Solution of the Chemical Master Equation", in the *Proceedings of the 8th International Conference on Computational Methods in Systems Biology, (CMSB 2010)*, ACM, 2010.

5. F. Didier, T.A. Henzinger, M. Mateescu, and V. Wolf, "Fast Adaptive Uniformization of the Chemical Master Equation", in the *Proceedings of the first International Workshop on High Performance Computational Systems Biology (HiBi 2009)*, IEEE Computer Society, 2009; and in *Systems Biology, IET journal*, 4:6, 2010.

6. F. Didier, T.A. Henzinger and M. Mateescu, and V. Wolf, "Approximation of Event Probabilities in Noisy Cellular Processes", in the *Proceedings of the 7th International Conference on Computational Methods in Systems Biology (CMSB 2009)*, Lecture Notes in Computer Science 5688, Springer, 2009; and in the *Theoretical Computer Science journal*, 412, 2011.

7. T. A. Henzinger, and M. Mateescu, V. Wolf, CAV '09. "Sliding Window Abstraction for Infinite Markov Chains", in the *Proceedings of the 21st International Conference on Computer Aided Verification (CAV 2009)*, Lecture Notes in Computer Science 5643, 2009; and as "Solving the chemical master equation using sliding windows", together with R. Goel, in the *BMC Systems Biology journal*, 4:42, 2010.

8. J. Fisher, T. A. Henzinger, M. Mateescu, and N. Piterman, "Bounded Asynchrony: Concurrency for Modeling Cell-Cell Interactions" in the Proceedings of the First International Workshop on Formal Methods in Systems Biology (FMSB 2008), Lecture Notes in Computer Science 5054, 2008.

Coordinates

Models and Theory of Computation
School of Computer and Communication Sciences
Ecole Polytechnique Féderale de Lausanne (EPFL)
CH-1015 Lausanne

Email: maria.mateescu@gmail.com

Personal Details

Date of birth:	29 July 1981
Citizenship:	Romanian
Marital status:	Single