PhD Dissertation

International Doctorate School in Information and Communication Technologies

DISI - University of Trento

On Efficient Algorithms for Stochastic Simulation of Biochemical Reaction Systems

Vo Hong Thanh

Advisor:

Dr. Roberto Zunino

Università degli Studi di Trento

November 2013

Abstract

Computational techniques provide invaluable tools for developing a quantitative understanding the complexity of biological systems. The knowledge of the biological system under study is formalized in a precise form by a *model*. A simulation algorithm will realize the dynamic interactions encoded in the model. The simulation can uncover biological implications and derive further predictive experiments. Several successful approaches with different levels of detail have been introduced to deal with various biological pathways including regulatory networks, metabolic pathways and signaling pathways. The Stochastic simulation algorithm (SSA), in particular, is an exact method to realize the time evolution of a well-mixed biochemical reaction network. It takes the inherent randomness in biological reactions and the discrete nature of involved molecular species as the main source in sampling a reaction event. SSA is useful for reaction networks with low populations of molecular species, especially key species. The macroscopic response can be significantly affected when these species involved in the reactions both quantitatively and qualitatively. Even though the underlying assumptions of SSA are obviously simplified for real biological networks, it has been proved having the capability of reproducing the stochastic effects in biological behaviour.

Essentially, SSA uses a Monte Carlo simulation technique to realize temporal behaviour of biochemical network. A reaction is randomly selected to fire at a time according to its *propensity* by conducting a search procedure. The fired reaction leads the system to a new configuration. At this new configuration,

reactions have to update their propensities to reflect the changes.

In this thesis we investigate new algorithms for improving performance of SSA. First, we study the application of tree-based search for improving the search of a reaction firing, and devise a solution to optimize the average search length. We prove that by a tree-based search the performance of SSA can be sensibly improved, moving the search from linear time complexity to logarithmic complexity. We combine this idea with others from the literature, and compare the performance of our algorithm with previous ones. Our experiments show that our algorithm is faster, especially on large models.

Second, we focus on reducing the cost of propensity updates. Although the computational cost for evaluating one reaction propensity is small, the cumulative cost for a large number of reactions contributes a significant portion to the simulation performance. Typical experiments show that the propensity updates contribute 65% to 85%, and in some special cases up to 99%, of the total simulation time even though a *dependency graph* was applied. Moreover, sometimes one models the kinetics using a complex propensity formula, further increasing the cost of propensity updates. We study and propose a new exact simulation algorithm, called *RSSA* named after *Rejection-based SSA*, to reduce the cost of propensity updates. The principle of RSSA is using an over-approximation of propensities to select a reaction firing. The exact propensity value is evaluated only as needed. Thus, the propensity updates are postponed and collapsed as much as possible. We show through experiments that the propensity updates by our algorithm is significantly reduced, and hence substantially improving the simulation time.

Third, we extend our study for reaction-diffusion processes. The simulation should explicitly account the diffusion of species in space. The compartment-based reaction-diffusion simulation is based on dividing the space into *subvolumes* so that the subvolumes are well-mixed. The diffusion of a species between subvolumes is modelled as an additional unimolecular reaction. We propose a

new algorithm, called *Rejection-based Reaction Diffusion* (RRD), to efficiently simulate such reaction-diffusion systems. RRD combines the tree-based search and the idea of RSSA to select the next reaction firing in a subvolume. The highlight of RRD comparing with previous algorithms is the selection of both the subvolume and the reaction uses only the over-approximation of propensities. We prove the correctness and experimentally show performance improvement of RRD over other compartment-based approaches in literature.

Finally, we focus on performing a statistical analysis of the targeted event by stochastic simulation. A direct application of SSA is generating trajectories and then counting the number of the successful ones. *Rare events*, which occur only with a very small probability, however, make this approach infeasible since a prohibitively large number of trajectories would need to be generated before the estimation becomes reasonably accurate. We propose a new method, called *splitting SSA* (sSSA), to improve the accuracy and efficiency of stochastic simulation while applying to this problem. Essentially, sSSA is a kind of biased simulation in which it encourages the evolution of the system making the target event more likely, yet in such a way that allows one to recover an unbiased estimated probability. We compare both performance and accuracy for sSSA and SSA by experimenting in some concrete scenarios. Experimental results prevail that sSSA is more efficient than the naive SSA approach.

Keywords

Systems biology, Stochastic simulation, SSA, Tree-based SSA, Huffman tree SSA, Rejection-based SSA, RSSA, Rejection-based reaction-diffusion, RRD, Rare event simulation, Splitting SSA, sSSA.

Acknowledgments

I would like to thank my supervisor Roberto Zunino. I am indebted to him very much for his countless help and support. He always has time for me to listen and give me useful advice in my journey in science.

I would like to express my grateful thank to my family and friends for their support during my study. The most dearly acknowledgments are for my wife, Truong Thi Thanh Thao, who always gives me love and strength to overcome difficult time.

Contents

Chapter 1

Introduction

1.1 Biological modelling and simulation

Recent advances in molecular biology have been doubtlessly continuing and increasing our knowledge of biological systems. The detailed quantitative data produced allow to characterize, for example, the entire human genome sequence and its products [144]. However, genes, proteins and their interconnections alone are not sufficient to explain all the complexities of living organisms. A cellular system, in essence, is a dynamic system in which its functions are not controlled only by the network structure but also the dynamics of involving elements. Explaining how the molecular interactions and, at its best, the combination principles emerging to a specific cellular behaviour needs a system-wide perspective. The cell differentiation during the cell cycle is just an example. By changing the experiment conditions, e.g., initial conditions, stimulus, the resulted cells can be very different, even counter-intuitive patterns. This is due to the dynamic characteristics and non-linearity of this process. A system level analysis of biological systems is thus a promising approach to provide an insight explaining of biological phenomena.

Systems biology is an emergent research area as a combination of system theory and molecular biology. It takes into account the structure and dynamic interactions within the biological network with the aim to understand how these give rise to a specific behaviour at the system level, and ultimately, to develop new biological systems for useful purposes e.g., effective prevention and/or treatment of diseases (see e.g. [87–89, 173] and references therein).

The computational modelling and simulation plays an important role in the development of systems biology in twofold. First, it abstracts out a biological network in term of a *model*. The model encodes the temporal evolution of its *state* in a formal form. Second, it allows to visualize and to predict the causal-effect of the biological system in time through a computer simulation.

Essentially, a model is an effort to explicitly encode the knowledge of biological system in a precise form. Depending on features of the biological system under study, the model should include sufficient information for analyzing the system dynamics. For example, at a detail molecular modelling, the model should manage all the detailed information, e.g., velocity and/or position, of all molecular species. A whole-cell model, in contrast, should include only a description of all key cellular processes. A biological model, to some extent, is therefore just an abstraction of the real system; however, it is useful to formalize the understanding of the biological system. So, modelling provides an effective way to highlight gaps in knowledge of biological systems.

The temporal behaviour of a given biological model is then realized by conducting *in silico* experiments. The simulation results are compared against with real experimental data. The inconsistency will show a lack of knowledge in the model of considered biological system. Models which are validated can be used to discover indirect and hidden implications in the biological system, which sometimes are hard to perform in wet lab. For example, one can isolate some vital genes and observe in detail their behaviour in individual as well as in together by in silico experiments. This, however, is obviously impossible in wet lab since the cell in such condition may not survive or even not exist. The results produced by in silico experiments are used for hypotheses forming, and suggest new experiments. Thus, the predictive feature of computer simulation makes it extremely useful for doing quantitative analysis of biochemical systems.

The biological modelling and simulation further contribute to the design and implement. A component-based approach is more effective than building the entire system from scratch, which is often more error-prone. The well-understood models with detailed interacting behaviour are reused as basic building blocks in a large model. The substitutable feature of this approach provides an opportunity to reprogram cellular functions to serve for special purposes of biological research [160].

Summing up, biological modelling and simulation in the post-genomic era are becoming increasingly important. The knowledge of biological system is able to integrate into a model, and make testable predictions through simulation. In silico experiments, in this sense, are highly preferred in term of speed, ease and cost; however, it is also important to emphasize that in silico experiments cannot be considered as a substitution of real biological experiments. In silico experiments thus are used in complement to biological research.

1.2 The need and challenges for stochastic simulation

Different levels of modelling and simulation detail have been adopted to investigate the dynamics of biological systems. At higher coarse-grained level the deterministic approach, where the concentration of molecular species are considered, has the capability of predicting dynamic behaviour of biochemical systems. The application of deterministic approach often lies on the *law of mass action* which states the rate of a reaction is directly proportional to the concentration of reactant molecules [14, 102, 168]. The time evolution of a biochemical network is completely described by a set of ordinary differential equations (ODEs), which is generally referred to as *Reaction Rate Equation*s (RREs). Hence, the complete dynamic picture of the system, given an initial condition, can be constructed by an analytic and/or numerical method [10, 125, 155].

Furthermore, a lot of well-developed tools, e.g., stability and bifurcation analysis [150], metabolic control analysis (MCA) [53], have been introduced for analyzing the behaviour of ODE.

The law of mass action has been successful to model chemical reactions at equilibrium (see [42, 73] for examples); however, its underlying assumption is obviously oversimplified for biological systems. The changes in population of molecular species due to reaction firings are assumed to be less significant so that population of molecular species are considered as continuous. The fluctuations of involved species, in this sense, have a negligible effect to the macroscopic trend of the molecular concentrations. Thus, the law of mass action describes only average behaviour. The molecules involved in biochemical reactions, however, are obviously discrete. Furthermore, it is common to find in a model few specific species, e.g., genes, mRNAs, which play a key role, yet have a very small population. Small changes in these species can lead to a significant quantitative and qualitative fluctuation in the behaviour of the overall biological system. Second, a collision between molecular species to form a reaction is inherently random. The occurrence of a random reaction can give rise to unexpected responses, e.g., bistability response pattern. Such random fluctuations at molecular level are inevitable and referred to as biological *noise*. The important of the fluctuations and noise in biological systems have been repeatedly pronounced in recent research (see e.g., [9,46,108,109,127,158,166]). The random effects in such systems can help to explain many biological phenomena, e.g., phenotypic variants [131]. Finally, biological noise itself has an important role in enhancing inter- and intra-cellular functions. The noise is propagated from cell to cell to modulate and improve the cellular signaling [122, 130]. A quantitative understanding of biological responses taking account of stochastic effects is preferred.

At molecular level, the molecular dynamics (MD) [3, 143], where the motions and interactions between molecules are governed by physical forces, is the most detailed and accurate method. It has to keep tracking all the positions, velocities as well as possible collisions of every molecules in the biological system. Although this approach yields an accurate result, it requires a very detailed knowledge of the molecules both in time and space, and computationally intensive in performing simulation. Hence, MD is limited to simulate the system only at the nanoscale of time and/or space. The *stochastic kinetics* is a more practical approach that still could capture the stochastic noise. In stochastic kinetics, the system state is denoted by a vector of population of species. Species can interact through coupled biochemical reactions. A reaction firing will cause the system state to move to a new state.

The stochastic kinetics is underpinned on that the probability a reaction firing in the next infinitesimal time can be expressed by a *propensity* function. In [60] a derivation for the existence of such propensity function for the so-called *elementary* reaction, which involves at most two molecular species as reactants, is provided. The dynamic time evolution of the reaction network thus can be described as a (continuous) jump Markov process. A complete mathematical form for expressing the time evolution of the system state is generally referred to as *Chemical Master Equation* (CME) [64]. A directly analytic solution of CME, however, is hard to obtain unless the system is very small. Fortunately, we can construct an exact realization of CME through a simulation method called *stochastic simulation algorithm (SSA)* [60, 61, 65]. SSA realizes a possible state transition by randomly selecting a reaction to fire according to its propensity. At the new state, affected reactions have to update their propensities to reflect the changes.

SSA, however, is often very computational demand for simulating large biological systems. In practice, large models are needed to investigate the noise effects to the whole regulatory system [150]. For example, one can observe the propagation of noise in a pathway and its impacts on the cell fate. The understanding of these effects is necessary for developing an automatic system design and control. The simulation time of a SSA run is mainly dominated by two sources: search for the next reaction firing and update the propensities after a reaction fired. First, an inefficient search for the next reaction firing such as the linear search is asymptotically increasing with the number of reactions in the model. The linear characteristic thus limits the application of SSA to large models. Second, a large model is typically encompassed with a large number of interconnections and (feedback) loops. The propensity updates required anytime the population of involved species is changed are also a computational bottleneck. Moreover, sometimes one models the kinetics using a complex propensity formula, e.g., the Michaelis-Menten equation, the Hill equation, further increasing the cost of propensity updates. The computation cost of SSA is further increased when relaxing the underlying assumptions of SSA. For example, to handle the movement of species in space, the extension of SSA is introduced by dividing space into subvolumes. A species can locally interact with other species inside a subvolume or jump to its neighbors. The search and update of reactions obviously take more computational demand because the number of species and reactions grow with the number of subvolumes.

Due to the stochastic behaviour in a single realization, a lot of simulation trajectories are required to ensure correct statistical information of the final reachable states. For example, to estimate the reaching probability of a given set of targeted states, one needs to generate an ensemble of independent SSA simulations (say 10^6 runs) and count which hits the target to collect a reasonable statistics. SSA will soon become inefficient to estimate the rare event probability since a prohibitively large number of trajectories, and of course very high computational effort, would need to be generated before the estimation becomes reasonably accurate.

In addition to these general characters, a biological model can exhibit multiscale behaviour. The reactions are often separated by different time scales in which some fast reactions will occur at a rate greater than other reactions.

In particular, in *stiff* system, the fast reactions occurs frequently and drive the system into stable state very fast. After this short fluctuation time, the slow reactions will determine the system dynamics. However, most of the time the simulation samples the fast reactions which is not the expected behaviour of the system. Furthermore, the population of some species involved in reactions may also many orders of magnitude larger than others. The fluctuations of these species, when reactions fire, are less significant. Keep tracking single reaction firings for large population species by SSA is obviously less efficient since a coarse-grained simulation method can be applied without loss of total simulation accuracy. Because of the inherent dynamics in biochemical reactions, a model can combine and mix all of these aspects in a very complicated manner. For example, the system exhibit stiffness at beginning, but then requires to consider a single reaction firings. It also can start with large population of some species then their population become small because of many reactions firings. These issues raise a great challenge for developing and implementing of an efficient stochastic simulation method [142, 154].

1.3 The objective of the thesis

In this thesis we aim to improve the existing methods and investigate new algorithms for efficiently performing exact stochastic simulation. We contribute to the improvement of SSA in following aspects:

- We study the effect of the **search** for the next reaction firing to the performance of SSA. We contribute to the improvement of SSA by proposing a tree-based search approach. We show, both in theory and in practice, that by using an underlying tree data structure to store reaction propensities the simulation time can be sensibly improved. Second, we predict the shape of the tree leading to optimal average search time. This turns out to be the Huffman tree, a well-known device used for data compression. Then, we study efficient approaches to rebuild the tree when it becomes non-optimal.

- We study the effect of the propensity **updates** to the overall performance of stochastic simulation. Even though a dependency graph can reduce the propensity updates to be model-dependent, in which only locally affected reactions have to recompute their propensities, still there are models, e.g., highly coupled reactions, where costly updates are required. The update cost is further increased if a complex propensity function is exploited to model complex effects, e.g., the allosteric effect in modelling protein binding mechanism. The simulation time is significantly affected by propensity updates. We propose a new algorithm, called RSSA, to avoid fully recomputing propensities of affected reactions as much as possible. RSSA uses an over-approximation of propensities to select a candidate reaction. The candidate reaction is then subjected to a rejection-based procedure to decide either accept this selected reaction to fire or (with low probability) reject it. We experimentally study different search procedures for finding a candidate reaction and discuss which leads to better performance, for different network sizes. We subsequently study several strategies for controlling the amount of over-approximation (hence, indirectly the acceptance probability), and analyze their impact to the simulation performance. We also discuss how to systematically optimize the tunable parameters of RSSA so to maximize its performance.

- We study the spatial effects in biological reactions. Although diffusion of species in space is inevitable, it is less significant when the diffusion time is many orders faster than the reaction time. The biological system, however, will exhibit spatial heterogeneity if this condition is violated. SSA has been extended to incorporate diffusion by dividing the space into well-stirred subvolumes. Species can locally interact in a subvolume or diffuse between subvolumes. The diffusion of species is modelled as first-order reactions. As a result, the number of species and reactions in a reaction-diffusion model are increased linearly with the number of subvolumes. The simulation thus requires a prohibitive computational cost for both of the search of a reaction firing in a subvolume and the update of affected reactions and subvolumes after a reaction fired. We contribute to this topic by proposing a new method called *RRD*. RRD combines the tree-based search and the principle of RSSA to improve performance of the stochastic reaction-diffusion simulation. First, a candidate subvolume is selected through a binary search on an over-approximation of subvolume propensities. Then, a candidate reaction in this subvolume is retrieved by using a fast lookup search on an over-approximation of reaction propensities. A rejection-based procedure is finally applied to either accept the reaction to fire or reject it. These features of RRD make it scale well with both large numbers of subvolumes and reactions.

- We study the problem of performing a statistical analysis of a targeted event of interest on a biological model. A large number of SSA runs may be required to achieve reasonable statistical accuracy of the event under study. The task becomes increasingly harder when considering *rare events*, which occur only with a very small probability. The estimated rare event probability produced by SSA may even be inaccurate. We contribute to this study by proposing a new method, called *sSSA*, to efficiently estimate the probability of a rare event. sSSA estimates the probability of a rare event through a kind of biased simulation. The state space is *split* into subsets defined so that the event becomes more likely to reach when moving from one subset to another. Hence, the simulated trajectories are gradually "pushed" towards the rare event following such subsets. The (unbiased) probability for the rare event is then estimated by counting the successful (biased) trajectories, and then applying a correction factor so to account for the bias.

1.4 Structure of the thesis

The outline of the thesis is the following.

In chapter 2 we briefly review modelling techniques to represent a biochemical reaction network. Then, we give a detailed review of stochastic simulation techniques, including exact, approximate and hybrid methods to improve the performance of SSA. The extensions of SSA obtained by relaxing its underlying assumptions i.e., reactions with delayed time and spatiality, are also reviewed.

In chapter 3 we describe in detail the application of tree-based search to improve the search of next reaction firing. The underlying data structure and algorithm for performing binary search are detailed. Then, we study which tree structures leading to an optimal search length and tree rebuilding strategies when the tree becomes non-optimal. A part of this chapter has been published in [156], of which an extended version is submitted for publication.

In chapter 4 we present key steps of RSSA for finding a reaction firing with its firing time based on the over-approximation propensities. We provide a formal proof for the correctness of RSSA. Then, we discuss different search procedures for finding a candidate reaction supported by RSSA as well as several mechanisms to control the amount of approximation, hence controlling the acceptance probability. A part of this chapter has been submitted for publication.

In chapter 5 we will describe in detail the RRD algorithm. The key steps for selecting a subvolume and a reaction firing in that subvolume are presented. A proof for correctness of RRD is also presented.

In chapter 6 we formulate the problem of rare event probability estimation in the stochastic simulation setting. Then, we present the sSSA algorithm and its features for improving the efficiency and accuracy of estimating the probability of rare events. A part of this chapter has been published in [157].

The conclusions and further research are in chapter 7.

Chapter 2

Stochastic Simulation: A Literature Review

2.1 Introduction

Molecular species, e.g., genes, mRNAs, proteins, are constantly moving inside a cell. Following a species trajectory, it can collide with other species. A collision between molecular species will form a reaction if it satisfies some specific conditions, e.g., activation energy, which are known as the *reaction kinetics*. The rate of a reaction, in essence, depends on a rate constant and reactants. The result of a reaction is new molecular species produced to help performing necessary activities of the cell. The reaction pathway is an organized reaction network to perform special cellular purposes. A biological system exploits different pathways by many mechanisms, e.g., feedback and feedforward loops at different levels, e.g., time and/or space to control, regulate and coordinate operations between cells. The understanding of these mechanisms becomes more difficult when random noise, yet important, is taken part in these processes. The stochastic framework provides promising tools for performing an insight analysis of the system behaviour at system-wide level.

Two important factors have to be established for the success of a stochastic approach. First, a modelling formalism should allow to encode the knowledge of the reaction network as well as its parameters in a more formal, precise and testable form. It must be simple, flexible and scalable enough for modelling different types of reaction networks ranging from very small, e.g., simple gene expression, to very large, e.g., complex signaling pathways, metabolism or even living organisms. Further, the model should be standardized so that it is able to share information, data and knowledge between communities. Second, a simulation algorithm is built to visualize the time evolution of the system. The simulation should be able to capture important features in the dynamics of biological processes. It also takes into account biological noise as an important factor affecting the system evolution. Thus, the grand challenge in computational biology is to model and simulate a full cellular organism [142, 154].

A lot of successful work has been established in literature to lay down the foundation for modelling biochemical reaction networks. A direct way to describe a reaction network is to write down the network as a list of coupled reactions. Modelling a reaction network by coupled reactions is simple and flexible. The network is easy to communicate between biologists and computer scientists. However, this modelling technique also has its own disadvantage. The number of reactions and their complex coupling in large models make it difficult to control. A graphical representation is an alternative modelling for reaction networks. For instance, a graph, e.g species-reaction graph, Petri net [171], can visualize the reaction network in a visual form. It thus unravels the hiearchical organization and causalities between components of the reaction network. Further, mathematical analysis on graph can be carried out to obtain a qualitative information about the dynamics of the network. Recent modelling formalisms, adapted from the computational area, have tried to improve the expressiveness of the model, e.g., π-calculus, state chart, discrete-event modelling (DEV) [37, 49]. They allow to explicitly represent the biological entities such as molecular species, reactions, as concurrent processes. Each process is an independent entity. It interacts and shares information with other processes concurrently through channels. A logical process could be implemented as an instance of a runnable process in a computer, so it is easy to turn an entire model into an executable simulation. Furthermore, these formalisms have strong and well-studied mathematical background. A lot of well-developed mathematical tools have been developed to support for useful analysis, e.g., checking equivalence behaviour, model checking [1, 113]. New modelling techniques such as rule-based modelling [50, 107, 146], also get more attention recently. They are introduced to overcome the explosion problem in modelling reaction pathways, e.g., signaling pathway. For example, in rule-based modelling, reactions are modelled as rules. A rule also encompasses with extra information for the reaction firing, i.e., reaction kinetics. If a rule is matched, the corresponding reactions is introduced to the system at runtime. Thus, all the possible reactions in the model do not need to be specified at the beginning of simulation.

Once the model developed, we can perform *in silico* experiments through a computer-based simulation. The dynamic interaction between species in the model can reveal indirect implications, unexpected behaviour which are complicated, unpredictable and even unknown at the modelling phase. The stochastic framework is often the choice to analyze random phenomena in biological responses. A reaction between molecular species is expressed as a stochastic process. The time associated with reactions is treated continuous, while the state is discrete, e.g., species population. The dynamics of the biological network thus can be expressed as a collection of stochastic equations. An analytic solution to these stochastic equations, however, is limited to small models only. Mathematical analysis is often intractable for large models. Stochastic simulation is an alternative approach to realize the dynamic behaviour of the given reaction network. A sample trajectory of the system is generated by sampling a possible reaction event. Thus, usually many trajectories should be generated in order to have a sufficient information about the system behaviour. Throughout the time, many simulation algorithms and software tools have been developed for performing biochemical simulation. These algorithms can count for the stochasticity in time and/or space.

For a well-mixed biochemical reaction system, the stochastic simulation algorithm (SSA) [60, 61] is a *de facto* standard for numerically sampling the time evolution of a biochemical reaction network. The development of SSA has the mathematical background on the chemical master equation (CME) [64], which completely describes the probability distribution of all possible state transitions. SSA takes into account the inherently random fluctuation of the involved molecular species as a main source in selecting a reaction firing. It is an exact method in the sense it does not introduce any source of approximation in selecting the reaction. In other words, it gives the same result as the analytic solution of CME, while the later is intractable for many cases. Essentially, SSA searches for a reaction to fire at a time based on a probability function. The reaction probability distribution depends on the (current) system state and the chemical kinetics. Anytime a reaction fires, the system configuration, i.e., the system state, as well as the reaction probability distribution have to be updated.

SSA often requires very computational demand for large models. In practice, a large model is needed to address and understand the regulatory affects to the cell behaviour. Several improvements to SSA has been introduced during the time course to make it applicable for large models. For example, to speed up the search of the next reaction firing, reactions is rearranged so that a reaction having higher probability is placed near the search position. For updating reactions, a *dependency graph* [59] is often exploited so that only locally dependent reactions should have to be updated. The update is thus reduced to be model-dependent. Therefore, for loosely coupled reaction networks, e.g., a linear chain, the update is only a constant factor. Some algorithms even sacrifice its exactness to achieve a higher performance. The main idea of approximate algorithms is trying to fire as many as possible the number of reactions, but still constrained the approximation by an error constraint. The most notable approximate method is the τ-leaping [63] algorithm. Although approximate algorithms indeed run faster, they expose serious problems especially to models having just some species at very low copy numbers. Firing many reactions in one time step yields the negative population for these species which is obviously infeasible in real experiments. A promising approach to solve this problem is the hybrid simulation [121]. It treats the system by two complementary parts. The part with low population species is simulated by an exact stochastic simulation, while the part with the high population species is treated by a fast simulation algorithm e.g., ODE integration, τ-leaping. Hence, it still achieves a better performance and also captures the important stochastic effects.

The assumptions of SSA, e.g. instantaneous reaction firing, well-mixed solution, is restricted for living cells. The effects of these factors when considering can alter the behaviour of the biological network significantly. Hence, SSA should be adapted to account for these factors. For example, the highly localization of species which is generally referred to as the *molecular crowding* [33,82] enhances the availability of species, and thus speeding up the operations of cellular processes. It also helps to explain important effects in biological systems, e.g., the excluded volume effect. Thus, taking spatial information into the stochastic simulation is a crucial task [153]. A possible extension of SSA for spatially heterogeneous environment is dividing space into well-mixed subvolumes. The diffusion of a molecular species between subvolumes is explicitly modelled by an additional unimolecular reaction. The extension of SSA in this manner is known as the *compartment-based* simulation.

In the following, we review the model representation techniques used to represent of biochemical reaction networks. Although the modelling of biochemical systems is attractive and has been continuously increasing, a thorough review is out of scope of this thesis (see e.g., [37, 49, 50, 107, 146, 171] and references therein for more discussion). In the review, we focus only on the modelling formalisms that we directly apply for developing of our simulation algorithm. Then, we are going to details of the algorithms for conducting stochastic simulation of biochemical reaction networks. We cover fundamental ideas of SSA as well as efficient formulations proposed during the time course. We also present a brief review of approximate and hybrid methods to improve the performance by the cost of its exactness. The extensions of SSA by relaxing the underlying assumptions of the biochemical reaction networks are also reviewed. Two possible extensions are reviewed namely: reactions with delays and reactions with spatiality.

2.2 Reaction network representation

2.2.1 Coupled reaction list

Listing all the reactions in the network is a direct way to specify reactions of the model. The network thus will be expressed in form of coupled reactions. Let consider a biochemical reaction system consisting n species denoted as $S_1, ...S_n$. These species interact through m reactions $R_1, ...R_m$. Each reaction has the following general form:

$$R_j : v_{1j}S_1 + ... + v_{nj}S_n \overset{k_j}{\rightarrow} v'_{1j}S_1 + ... + v'_{nj}S_n \tag{2.1}$$

where v_{ij} and v'_{ij} are referred to as *stoichiometric coefficients*. In fact v_{ij} is the number of species S_i are consumed and, in contrast, v'_{ij} is the number of species are produced by reaction R_j. In this general reaction form, we allow some species to appear in the both side of a reaction. The appearance of such species is only to increase the rate of the reaction and this species is generally called a *catalyst*. k_j is the (stochastic) *rate constant* of reaction R_j. A reversible reaction in this representation should be expressed explicitly. The reversible reaction is thus considered as two separated irreversible reactions, and they are treated independently.

A coupled reaction list intuitively shows the coupling of the species in the model. It itself can give a qualitative structure of the system. Because of the simplicity and flexibility of the representation one can easily add, modify and remove reactions to extend the model. This modelling has been widely accepted to represent a reaction network. The systems biology markup language (SBML) [54, 77, 78, 152] is an attempt to standardize the modelling process with the help of a computer software. SBML encodes the reaction list in an independent format (the XML format). Thus, the model is easy to store, transfer and parse by a software component. There are also similar approaches to ease the modelling of reaction list with the help of computer e.g., CellML, BioPAX [44, 92, 104].

The coupled reaction list, however, also has its own disadvantages. First, a practical model often contains a lot of reactions. the model becomes extremely complex and even uncontrollable when modelling large networks. Second, it does not support for structural analysis. This preliminary analysis can give a substantial information for guiding the simulation development. This information is also useful in understanding the system dynamics at runtime. Furthermore, because the reaction model is not associated with necessary information, i.e., reaction kinetics and initial condition, it has to be tailored with this information before it can be simulated.

2.2.2 Graphical network diagram

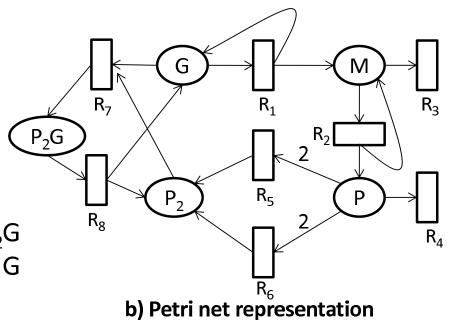
A graphical representation is a visual approach to model a biochemical reaction network. It contains the same information as a coupled reaction list, but presents the reactions in a diagrammatic format. Thus, it is easy to understand the hierarchical organization of the reaction network. Because a graphical model is backed on a rigorous mathematical structure, i.e., a discrete graph, several well-developed tools in this area can be applied to support for analyzing the organizing structure of the reaction network. The structure information briefly characterizes the dynamic behaviour of the corresponding biological system. The development of a graphical model with the help of computer, e.g., JDesigner, JigCell, make it become more easier. Recently, an effort to make the standard notations for network diagrams using the system biology graphical notation (SBGN) is proposed [81, 119], hence enhancing the quality and the usability of models.

The species-reaction (SR) graph is a natural representation of a biochemical reaction network. It is a type of bipartite graphs where nodes are completely divided into two types: the species nodes and the reaction nodes. A species node represents for a molecular species involved in the model, while a reaction node denotes for a reaction between species. A directed edge from a species node to a reaction node indicates that the species is a reactant of the reaction. In contrast, an edge from a reaction to a species indicates that the corresponding species is a product of the reaction. The edge between a species node and a reaction node is further attributed with a weight. This value denotes the stoichiometry of the species in the reaction.

A Petri net [69, 124, 133] is an another graphical modelling of the biochemical reaction network, but is augmented with rigorous mathematical semantic rules. Thus, it takes advantage over the SR graph. The Petri net is grounded also on a directed bipartite graph in which a species node is called a *place*, and a reaction node is called a *transition*. The place is associated with a number of *token*s, which are the population of the corresponding species. A configuration of the tokens in places at a time is referred to as a *marking*. When a transition fires, corresponding with a reaction fires, the tokens in the paces are redistributed. The system then moves to another marking. A transition firing is able to be modified to account for the random noise. Furthermore, properties of the model encoding in the Petri net such as network invariants e.g., P- and T-invariants, reachability, can be derived to support the simulation analysis. Thus, the Petri net is very well-suited for stochastic modelling and simulation.

R_1: $G \rightarrow G + M$
R_2: $M \rightarrow M + P$
R_3: $M \rightarrow \_$
R_4: $P \rightarrow \_$
R_5: $2P \rightarrow P_2$
R_6: $P_2 \rightarrow 2P$
R_7: $P_2 + G \rightarrow P_2G$
R_8: $P_2G \rightarrow P_2 + G$

a) Reaction list



b) Petri net representation

	reactant					product				
	G	M	P	P_2	P_2G	G	M	P	P_2	P_2G
R_1	1	0	0	0	0	1	1	0	0	0
R_2	0	1	0	0	0	0	1	1	0	0
R_3	0	1	0	0	0	0	0	0	0	0
R_4	0	0	1	0	0	0	0	0	0	0
R_5	0	0	2	0	0	0	0	0	1	0
R_6	0	0	0	1	0	0	0	2	0	0
R_7	1	0	0	1	0	0	0	0	0	1
R_8	0	0	0	0	1	1	0	0	1	0

c) Stoichiometric matrix

Figure 2.1: The gene expression model is represented by a) a coupled reaction list, b) a Petri-net and c) the corresponding stoichiometric matrix

To store the underlying bipartite graph of a graphical model in a computer, we make use of a matrix. An element in the matrix is corresponding with an directed edge between two nodes. The corresponding element is set with a value is the weight (stoichiometry) of such edge. Such matrix is generally referred to as the *stoichiometric matrix*. Since the matrix is often sparse (with many zero elements), we can apply the sparse matrix computation techniques to reduce its size and processing time. The figure 2.1 gives an example of different representations for the gene expression model.

2.3 Simulation algorithm

2.3.1 Exact stochastic simulation

Let consider a well-mixed biochemical reaction system. The cell is assumed to be fixed to a constant volume, and is in a thermal equilibrium. The position and

Table 2.1: Propensity function for elementary reactions

R_1: $\emptyset \xrightarrow{k_1}$products	$a_1 = k_1$
R_2: $S_i \xrightarrow{k_2}$products	$a_2 = k_2 \cdot X_i$
R_3: $S_i + S_j \xrightarrow{k_3}$products	$a_3 = k_3 \cdot X_i \cdot X_j$
R_4: $S_i + S_i \xrightarrow{k_4}$products	$a_4 = k_4 \frac{X_i(X_i-1)}{2}$

speed of molecular species in the cell volume, by these assumptions, become randomized. In fact, they are randomly distributed following the thermodynamics law. We therefore only need to consider the population of molecular species, while ignoring all the positions, velocities of species. Let $X_i(t)$ denote the population of species S_i at time t. Thus, the state vector $X(t)$ of the system at time t is represented by a n-vector $X(t) = (X_1(t), ..., X_n(t))$.

The change of the system state at time $t + dt$ which is the consequence of the next reaction R_j firing is denoted by a state change vector v_j. Note that v_j is corresponding to a row of the stoichiometric matrix. Thus, the state transition of the system is formulated as:

$$X(t + dt) = X(t) + v_j \tag{2.2}$$

The quantity characterizing the probability reaction R_j firing is termed a propensity function a_j. It is defined so that $a_j(x)dt$ is the probability reaction R_j will fire in the next infinitesimal time $t + dt$ given the current state $X(t) = x$ at time t. This is referred to as the *fundamental hypothesis* [60] of the stochastic kinetics simulation. A physical derivation for the existence of such propensity function for the *elementary reaction*s is provided in [60, 111]. We summarize the form of these formulas in the following table 2.1.

By the fundamental hypothesis, the biochemical reaction system can be modelled as a (continuous-time) jump Markov process. Let $P(x, t)$ be the probability of system being in state $X(t) = x$ at time t. The differential equation expresses the complete time evolution of $P(x, t|x_0, t_0)$ with initial state

$X(t_0) = x_0$ at time t_0 given in Equ. 2.3.

$$\frac{\partial P(x,t|x_0,t_0)}{\partial t} = \sum_{j=1}^{m} \left[a_j(x-v_j)P(x-v_j,t|x_0,t_0) - a_j(x)P(x,t|x_0,t_0) \right] \tag{2.3}$$

Equ. 2.3 is generally called the chemical master equation (CME). It completely determines the time evolution of the system at any particular time t. CME is indeed a collection of differential equations describing all the state transitions by biochemical reactions. The number of equations in CME is thus increasing exponentially with all possible state transitions. For example, let consider a system where each species has only two states: 0 and 1. For n species we will have total 2^n equations. A full analytic solution of CME is obviously intractable for most of practical problems where n is large enough. Some recent computational approaches [116,172] have tried to solve CME directly but at the cost of an approximation error. In this thesis, we exploit the simulation technique to sample the possible solutions of CME instead. The simulation realizes a trajectory of the system evolution by sampling the *next reaction probability* density function $p(\tau, j|x, t)$, in which $p(\tau, j|x, t)d\tau$ is the probability a reaction will be fired in the next time $t+\tau+d\tau$ and it is the reaction R_j, provided that we are in state $X(t) = x$. The next reaction probability is indeed a joint probability of the firing time τ and the selected probability of reaction R_j. We have:

$$p(\tau, j|x, t)dt = a_j(x)exp(-a_0(x)\tau)dt \tag{2.4}$$

where

$$a_0(x) = \sum_{j=1}^{m} a_j(x) \tag{2.5}$$

while τ and j are the time of the reaction firing and its index, respectively.

The Equ. 2.4-2.5 is the basis for the stochastic simulation algorithm (SSA). It imposes two important things. First, the firing time is exponential distributed with mean $1/a_0$. Second, the probability reaction R_j is selected to fire at that time is a discrete probability mass function a_j/a_0. There are two implementations of SSA which have the same stochastic behaviour were introduced. They are known as the Direct Method (DM) and the First Reaction Method (FRM).

DM directly computes the reaction firing time τ by inverse the exponential distribution $a_0 exp(-a_0(x)\tau)$, and then searches for reaction R_j to fire according to its probability a_j/a_0. DM requires two random number for doing a simulation step. Let r_1 and r_2 be random numbers generated from a uniform distribution $U(0, 1)$. The first number is used to compute the firing time τ, while the second one is used to decide which the reaction R_j fires at that time.

$$\tau = \frac{1}{a_0(x)} ln \left(\frac{1}{r_1} \right) \tag{2.6}$$

$$j = \text{the smallest } j \text{ s.t. } \sum_{k=1}^{j} a_k(x) > r_2 a_0(x) \tag{2.7}$$

The search for a reaction firing R_j in DM is directly implemented by continuously accumulating the sum of propensities on-the-fly until it satisfies the condition $\sum_{k=1}^{j} a_k(x) > r_2 a_0(x)$. It is equivalent with a linear search.

Having the time τ and the fired reaction R_j, DM jumps current system state to the new state $x + v_j$, and updates current time to the new time $t + \tau$. The propensities of reactions are updated to reflect the change in the system state as well. The simulation will loop until the current time is passed over a predetermined simulation time T_{max}. We briefly outline the DM algorithm in Alg. 1 for the ease of reference.

The key point of the DM algorithm is the propensities $a_j(x)$s are computed once at the start of the simulation, and then updated as soon as the state x changes. In Alg. 1 all the reactions have to update their propensities after a reaction firing. The update step is obviously inefficient with a large model. To speed up the propensity updates, it is common to exploit a dependency graph between reactions, which describes which propensities actually need to be recomputed after every reaction firings. In other words, only locally affected reactions

Algorithm 1 Direct Method (DM)

```
 1: initialize system time $t = 0$ and system state $x = x_0$
 2: while $t < t_{max}$ do
 3:     for all reaction $R_j$ do
 4:         compute $a_j$
 5:     end for
 6:     compute $a_0$
 7:     generate two random numbers $r_1, r_2 \sim U(0, 1)$
 8:     set $\tau = 1/a_0(x) ln \left(\frac{1}{r_1}\right)$
 9:     search for the next reaction $R_j$ by continuously accumulating propensities $a_j$ until
        $\sum_{k=1}^{j} a_k(x) > r_2 a_0(x)$
10:     update the time $t = t + \tau$ and system state $x = x + v_j$
11: end while
```

have to be recalculated their propensities. The dependency graph $DG(V, E)$ is a directed graph (see Fig. 2.2 for an example) which contains the reactions as vertices V, while an directed edge $e(R_i, R_j) \in E$ if and only if $R_j \in$ affects(R_i), the set of reactions affected by R_i. Formally

$$\text{affects}(R_i) = \{\ R_j\ |(\text{reactants}(R_i) \cup \text{products}(R_i)) \cap \text{ reactants}(R_j) \neq \emptyset\} \quad (2.8)$$

where reactants(R_i) and products(R_i) are the set of species taking part in reaction R_i as reactants and products, respectively. Because a directed catalyst is not consumed by the reaction itself, it is excluded from the reactants and products of the reaction. Hence, by the dependency graph update mechanism, the propensity updates are now reduced to be model-dependent.

FRM is mathematically equivalent with DM but proceeds in a different manner. It is a type of racing algorithm. The reaction with smallest putative time is selected to fire next. Thus, in each simulation loop, m random numbers $r_1 \ldots r_m \sim U(0, 1)$ are used to generate the putative times of reactions. The putative time τ_j of reaction R_j is computed as:

$$\tau_j = \frac{1}{a_j(x)} ln \left(\frac{1}{r_j}\right), j = 1 \ldots m \quad (2.9)$$

No.	Reaction
1	$A + B \longrightarrow C$
2	$C + B \longrightarrow D$
3	$E + C \longrightarrow 2C + F$
4	$2C \longrightarrow G$
5	$F + H \longrightarrow B$

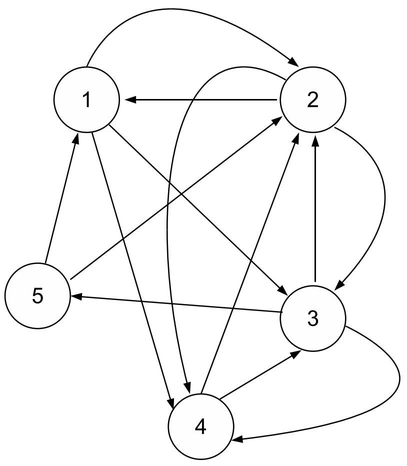


Figure 2.2: Dependency graph (removing self affected edges)

The reaction R_j having the smallest putative time $\tau_j = \min(\tau_1, \ldots, \tau_m)$ is selected to fire. The propensity update in FRM is done similar to DM. The mathematical equivalence between FRM and DM is derived directly from the property of the exponential distribution [60].

FRM takes m random numbers in each simulation step to compute the putative times of reactions. But, only one is actually consumed by the simulation, while $m - 1$ random numbers are discarded. A lot of random numbers waste while applying to large models. FRM is thus less efficient and often runs slower than DM. However, treating each reaction as a separated process, FRM allows to consider in detail the effects of each reaction to the overall system dynamics. For example, we can easily modify the propensity of a reaction taking into account the effect of e.g., cell size changing during the simulation time. This is known as the *random-time change* representation [97]. The firing time of a reaction can even be modelled by different distributions, e.g., the Erlang, the Hyperexponential distribution [117]. Second, FRM allows to see the simulation as a discrete-event simulation algorithm. And, there are many efficient event-queue data structures [20, 68, 83, 135] developed in computer science so that they can be directly applied to improve the performance of FRM. The most notable improvement of FRM as a discrete-event simulation is the Next Reaction Method (NRM) [59]. NRM uses a special priority queue, called the binary heap, to store the putative reaction times. Retrieving the smallest putative time is constant since it is always on the top of the heap. After a reaction is selected to fire, NRM has to maintain the priority queue to reflect the change in the system; however, it does this in a clever way. NRM exploits the scaling properties of the exponential distribution and dependency graph to improve the propensity updates. By this way, the absolute putative time has to be used, instead of relative putative time in original FRM. There are two cases the computing of new putative times and maintaining the heap are required. In the first case, the reaction that has to update its propensity is itself the reaction firing. The new reaction propensity is evaluated. Then, the new putative time is generated following Eq. 2.9. In the second situation, the reactions are dependent reactions (the affected reactions in the dependency graph). The scaling property of exponential distribution will be exploited to scale up their putative times. Assuming that the system moves from the state x to the new state x^{new} with the firing time t. Let τ_j^{new} be the new putative time of reaction R_j at this new state. It is scaled as $\tau_j^{new} = (a_j(x^{new})/a_j(x))(\tau_j - t) + t$. So, we do not need to generate additional random numbers for updating the putative times of affected reactions. There only one random number is required for each simulation step. This would save a lot of computational resource as the number of reactions m is large. In fact, the complexity of a call to binary heap consolidation takes logarithmic time i.e., $O(\log(m))$. Thus, NRM, in worst case, takes logarithmic time for a simulation loop assuming a constant number of affected reactions in the model.

A software package called Moleculizer [105] exploits these two characters of FRM to design an efficient simulation for the intra-cellular biochemical systems, i.e., the pheromone signal transduction pathway in Yeast. Due to the complexities of receptor-binding mechanism the number reactions in the model is exponentially increasing. The reactions in entire network is possibly not able to introduce to the simulation at beginning. Moleculizer takes over this problem by introducing the species and reactions to the simulation only as needed. The propensity of new introduced reactions will be modified in consistency with physical properties of this reaction. The new introduced reaction event is then efficiently controlled by a simplified version of queue-event data structure in NRM.

Although NRM is often faster than FRM, DM, it also exposes challenges for implementing the complex data structure used. In some special classes of problems, the complex data structure even negates the performance of NRM. For example, in [29], it showed that the runtime of NRM is actually slower than DM when applied for highly coupled and multiscale reactions models e.g., the heat shock response model of E. Coli. In [29], it also introduces an formulation to improve the performance of DM. This new formulation is called the Optimized Direct Method (ODM). ODM improves the search of DM based on a careful observation that the searching of the next reaction firing will faster if propensities are sorted in descending order. Indeed, the constraint in Eq. 2.7 is faster to satisfy if we rearrange the propensities in a descending order. This new formulation will achieve a great speed up gain if the system contains disparate ranges of propensity values. In ODM, the order of propensity values is predicted by pre-run simulations. The average values of propensities are used as criteria for ordering the reactions. The Sorting Direct Method (SDM) [110] shares the same idea with ODM, but it uses a different technique to order the reactions. SDM dynamically bubbles the reactions instead. Anytime a reaction fires, its new propensity is computed. Its index is then exchanged with the next lowest propensity (if exists). The bubble step is also applied to all affected reactions. At the end, an order for reactions propensities is established without a pre-run simulation.

Sorting of reaction propensities does make the linear search of DM run faster.

It, however, potentially makes the search less accurate [65]. A truncation error can happen when the sum of the biggest propensities is represented by a fixed-size floating number. For example, consider a floating point number with k precision in a computer representation. If the propensity of a reaction is k orders of magnitude smaller than the sum of biggest propensities placed before it in the decreasing sorted order. This reaction is thus never selected to fire if a decreasing order of propensity values is used. The implementation of sorting of reaction should require an infinite precision number representation. However, the most restriction of linear search, even reactions are ordered, is its time complexity, in the worst case, is increasing linearly with the number of reactions m, i.e., $O(m)$. The search thus becomes very slow to as applied to large models.

There are several formulations have been proposed during time to reduce the complexity of the linear search used in DM. One possible approach is dividing the reactions into groups. The search is now composing of two consecutive steps. First, the group containing the next reaction is discovered. Second, the next reaction firing in the corresponding group is retrieved out. In [106], these two steps are done through two linear searches. The first search discovers the group based on the total propensity of each group. And, the second search retrieves the next reaction firing in corresponding group by its propensity. In [145, 147], the grouping of reactions is also exploited, but the search of the next reaction in group is implemented by an acceptance-rejection procedure. A group is associated with a constraint. More precisely, reaction R_j belonging to group k must satisfy the group constraint: $b^{k-1} \leq a_j \leq b^k$ where b is a selected base (e.g., $b = 2$ in [147]). Then, the search of reaction firing is done as follows. A standard linear search is conducted to find out a group k containing the next reaction. The next reaction within the group k is discovered by applying the *rejection* mechanism with the chosen hat function b^k. This formulation is referred to as the composition rejection SSA (CR-SSA). The complexity for the long run of CR-SSA in searching the next reaction firing is constant time. The assumptions for the constant time of CR-SSA are: 1) the number of dependent reactions of a firing reaction should be restricted to a constant factor, and 2) the reaction propensities which are varied by reaction firings are less significant. Once these assumptions are violated, CR-SSA will spend a lot of time adding and removing reactions to appropriate groups. The CR-SSA performance therefore can be very slow. This has been shown by experiments in [106].

If reactions are divided into groups so that each group contains only two reactions, the search of the next reaction thus needs only one comparison to discover the next branch in the search path. In this sense we have a binary search [18, 103, 156]. The binary search obviously achieves better performance than linear search, but it requires to pre-compute the partial sums of propensities. These values have to be stored in a tree structure so that we can apply the dependency-graph update mechanism. The time complexity of a tree-based search SSA is logarithmic both in search and update. We are going to the detail of the tree-based search on the next chapter. There we also predict and discuss the tree leading to the optimal search length.

Instead of grouping reactions, the partial propensity SSA (PSSA) [128] factorizes and groups the reactants. The reactions sharing the same reactants are grouped. Only the partial propensities related to a reactant are computed. PSSA then exploits a complex data structure to store reactants as well as partial propensities. An equivalent procedure with SSA to find the next reaction firing is proposed. The complexity of PSSA, in the worst case, is proportional with number of molecular species. PSSA therefore outperforms when applying to highly coupled reaction models. The current restriction of PSSA is that: 1) it only supports for reactions having at most two reactants, and 2) the reaction propensity is based mainly on the mass-action kinetics [128]. The key idea of reactant grouping and binary search to improve the performance of SSA also proposed in [80].

A different approach to improve SSA is discussed in [141]. It exploits the *uniformization* technique to improve the simulation performance. The idea of uniformization technique is using the upper-bound of total propensity to discretize the time. By the application of the upper-bound of total propensity, this approach introduces a dummy reaction, without changing the system state, to the current set of reactions. The rate of the dummy reaction is equal to the different between the upper-bound value and the current total propensity. Because the firing time of all reactions, including the dummy reaction, is all exponential distributed with the same mean corresponding to the inverse of the total propensity upper-bound, we do not need to generate the reaction firing time. Only the search of reactions and propensity updates are required. in order to approximate the upper-bound of total propensity it has to know a *global* upper-bound for the population of all species. This is hard to pre-compute. Indeed, even in the case such upper-bound is known, it may be several orders of magnitude larger than the actual total propensity e.g. if the system is stiff. In this case, simulation would spend a lot of time firing the dummy reaction, hence frequently following self-loops.

2.3.2 Approximate stochastic simulation

Essentially, an approximate method speeds up the simulation by sacrificing its accuracy. It tries to execute as many as possible the number of reaction events in one simulation step. This is the main different with SSA where only one reaction event occurs at time. There are many approximate methods introduced, see for example [62, 115, 134], in which the most notable algorithm is the τ-leaping method. The time axis in τ-leaping is divided into (small) time intervals. The changes of all reaction propensities in a time interval are considered less significant and assumed to be constant. This condition is known as the *leap condition*.

Let $[t, t+\tau)$ be a time interval in which the propensity of any particular reaction R_j satisfies the leap condition. In other words, the propensity $a_j(x)$ is remained essentially constant during that time interval. The number of times reaction R_j occurring is so a Poisson process $Poisson(a_j(x)\tau)$. Let k_j be the number of times reaction R_j fires during the time interval $[t, t+\tau)$. Thus, we have that $k_j \sim Poisson(a_j(x)\tau)$. Each occurrence of R_j causes the system state to change an amount $x + v_j$. So, the net change of the system state by firing k_j times reaction R_j in the time interval $[t, t+\tau)$ is $x + k_j \cdot v_j$. Based on this observation the τ-leaping is proceeding as follows.

The simulation time T_max is divided into time intervals $[t, t+\tau)$ so that the leap condition is satisfied on each interval. In each simulation step, m Poisson random numbers $k_j \sim Poisson(a_j(x)\tau)$ for all $j = 1 \ldots m$ are generated. The system state changing by m reactions firing in an interval are updated by:

$$X(t+\tau) = X(t) + \sum_{j=1}^{m} k_j v_j \tag{2.10}$$

The accuracy of the τ-leaping thus is strongly depending on the choosing of an appropriate τ value. In principle, a *post-leap* check can be applied. That is we start with an predefined arbitrary (small) τ value. Then, we check the difference in the reaction propensity after that leaf. If all the differences are acceptable (i.e., satisfying the leap condition) then the leaf is accepted. Otherwise, τ should be reduced. More precisely, let x and x^τ be the state before and after the leap τ. The absolute change in propensity of reaction R_j is computed and ensured to be sufficiently small comparing with an error parameter ϵ, i.e., $\|a_j(x^\tau) - a_j(x)\| < \epsilon$ for all $j = 1 \ldots m$. If the change in propensity of any reaction violates this condition, τ is reduced e.g., to a half, and the checking procedure repeats. The post-leap working in this manner, however, potentially biases the system away from large yet reasonable changes in the state.

The *pre-leap* is thus often more promising than the post-leap. It instead computes the leap τ by postulating the expected change in propensities at the new expected state. The expected change is calculated and checked against whether it is acceptably small. Several strategies have been introduced for doing the pre-leap check. In [62] the expected change in propensities is suggested to be bound by $a_0(x)$, i.e., $\|a_j(x^\tau) - a_j(x)\| \leq \epsilon a_0(x)$ for all $j = 1 \ldots m$ where $0 < \epsilon \ll 1$ is the error control parameter. This original idea of the leap selection is extended and improved by [26, 66]. In [27], a new leap selection procedure is proposed in which the relative change in propensity of a reaction is bound by its current propensity instead of total sum of propensities.

A subtle problem occurring in the τ-leaping is the negative population of species. The Poison random variable $k_j \sim P(a_j(x)\tau)$, in general, is unbound. The population of a species after the leap thus can get negative. It is obviously unrealistic and should be prevented during the simulation. Several solutions have been introduced to solve the negative population. In [32, 159] a Binomial distribution with the same mean with the Poison process $P(a_j(x)\tau)$ is used instead to sample the number of reactions events in a leap. The negative population is avoided because the Binomial random number is bound. In [123], it constraints the changes in species population by solving an integer linear programming problem. The number of reaction events is then sampled by a Multinomial distribution. An another solution to this problem is dividing the system in two parts [25]. The species which have the large population is put in safe part which can directly apply the τ-leaping method, while the low population species are put in the *critical region*. The reactions involving with critical species are treated in individual by e.g., SSA. For stiff systems, the leaf τ of an (explicit) τ-leaping selection is very small which is in the order of the inverse of the total propensity a_0. The implicit τ-leaping [132] handles this obstacle to allow to choose an arbitrary large τ value by applying an implicit approximation form. However, the state change vector now is not an integer vector. It has to be round off to the nearest integer. This introduces an additional source of approximation to simulation. A combination of explicit-implicit τ-leaping is proposed recently in [28].

In [11] the K-leap method and in [22] the R-leap method, respectively, are alternatives for the τ-leaping method. The advantage of these methods is the number of reaction firings during a leap is controllable. The negative population never happens, and thus improving the simulation accuracy. These methods are variants of k_α-leaping method proposed in [62]. The principle of these methods is the total number of reaction firings during a leap is predefined and constrained. The leap τ is proved to be following a Gamma distribution, while the number of times a reaction firings during a leap is following a Multinomial distribution. Then, several sampling techniques have introduced for both of these methods to generate a suitable τ value.

The τ-leaping is not only used for improving the performance of SSA, it but also bridges a connection to the deterministic simulation [65]. Let suppose the leap condition is relaxed so that τ is still small enough to satisfy the leap condition, but the expected number of reaction firings in a leap is also large enough, i.e., $a_j(x)\tau \gg 1$ for all $j = 1 \dots m$. By this new condition, the Poisson distribution is approximated by a Normal distribution with the mean and variance are $a_j(x)\tau$. Thus, Eq. 2.10 is rewritten by:

$$\begin{aligned} X(t+\tau) &= X(t) + \sum_{j=1}^{m} N_j(a_j\tau, a_j\tau)v_j \\ &= X(t) + \sum_{j=1}^{m} v_j a_j \tau + \sum_{j=1}^{m} v_j \sqrt{a_j} N_j(0,1)\sqrt{\tau} \end{aligned} \tag{2.11}$$

where $N_j(\mu, \sigma^2)$ denotes a Normal distribution with mean μ and variance σ^2. To derivation of Eq 2.11 makes use a special property in conversion of a Normal distribution to standard Normal distribution $N(0,1)$ i.e., $N_j(\mu, \sigma^2) = \mu + \sigma N(0,1)$.

The Eq. 2.11 is referred to as the chemical Langevin equation (CLE). The equivalent differential formulation of Eq. 2.11 is given in Eq. 2.12.

$$\frac{dX(t)}{dt} = \sum_{j=1}^{m} v_j a_j + \sum_{j=1}^{m} v_j \sqrt{a_j} \Gamma_j(t) \tag{2.12}$$

where $\Gamma_j(t)$ is an independent Gaussian white-noise process.

In the *thermodynamic limit,* where the volume size and the species population is increasing to infinity, but the species concentration (the ratio between species population and volume size) is kept roughly constant, the random fluctuation term $\sum_{j=1}^{m} v_j \sqrt{a_j} \Gamma_j(t)$ in Eq. 2.12 grows slowly (in square root) comparing with other terms (in linearity). This term is thus negligible small contribute to the macroscopic change of the system and can be ignored. In other words, the fluctuation in population of species in Eq. 2.12 is able to remove. Eq. 2.12 approximate to be:

$$\frac{d[X]}{dt} = \sum_{j=1}^{m} f([X]) \tag{2.13}$$

in which $[X]$ denotes the species concentration vector, and a function f presents the changes of the species concentration by reactions. The Equation 2.13 is the general form of RREs used in deterministic simulation. Hence, the stochastic approach in the thermodynamic limit converges to the deterministic one.

2.3.3 Hybrid stochastic simulation

Hybrid methods are proposed to efficiently simulate the system with a great disparity in the species population. The high population species are simulated with a less computational technique since the fluctuations in these species are less significant. The low population species will be simulated by an exact method so that it is still able to captures the significant fluctuations [16, 118]. The hybrid approach thus still achieves a better performance, but also reproduces the stochastic effects by the low population species.

The principle of the hybrid approach is dividing the system into two subsystems. These parts will be simulated by different simulation methods, but they are complementary to each others. An intuitive partitioning strategy is to partition reactions into subsets of fast and slow reactions. Mathematically, it is equivalent to partition CME. The fast reactions often, but not always, involves high population species. The rest will be called slow reactions. Two subsystems are assumed to evolve independently. The fast reactions is integrated by, e.g., an ODE solver. The slow reactions is simulated by an SSA variant to retain the important fluctuations. Because the slow reactions, in general, is dependent to the fast species, their propensities can change if a fast reaction fires. For this reason, the propensity of slow reaction have to modify to use the random time varying propensity.

For the success of a hybrid method, several aspects have to be considered. First, the criteria as well as their reliability are applied for partitioning of the system. Second, how the partition is done in static or in dynamic. Third, how the synchronization between simulation techniques i.e., between the deterministic vs. stochastic as well as the data conversion i.e., between the species concentration vs. population, continuous vs. discrete. Lastly, how to treat the fast reactions involving also the low population species.

There are several hybrid methods has been proposed in literature. We review three main approaches in the following.

- The ODE/SSA hybrid. In [2, 84] it proposed a combination of SSA and ODE solver to simulate the system. An ODE solver is used to integrate the high population species part, while an SSA variant simulates for low population species. The algorithm works by partitioning the species and reactions as well as choosing a fixed integration time step Δt for the ODE integration. Although, note that, Δt could be adaptively decided in modern ODE solver. The ODE/SSA hybrid is proceeding as follows. First, an ODE integration with the time step Δt is computed with assumption that there is no slow reaction event occurring. The time-varying propensities of the slow reactions in this time step are evaluated. Then, the firing time δt for a slow reaction event is derived. In particular case, the time step Δt is chosen small enough so that the changes in slow reaction propensities are assumed to be constant. The computing of the slow reaction event thus does not require to use the random time change technique and is greatly simplified. Finally, the simulation decides which event will update the system. If the slow reaction event is occurring before the ODE integration, i.e., $\delta t < \Delta t$, a slow reaction is fired. The fast species involved in this slow reaction is updated as well. In the other case, only the ODE integration takes place. A new simulation iteration is executed after that.

- The CLE/SSA hybrid. This hybrid simulation is a combination of a discrete simulation for slow reactions and a CLE solver for fast reactions [72, 140]. The partitioning of reactions is treated dynamically. A reaction is considered to be fast if it satisfies the conditions 1) $a_j \Delta t \geq \lambda$ and 2) $x \geq \epsilon \left|v_j\right|$ in which Δt is the time step for updating the fast reactions, λ and ϵ are parameters to control the partitioning. For example, in [140], λ and ϵ are assigned to be 10 and 100, respectively. During the time course if a fast reaction violates the partitioning condition, it is automatically moved to slow reaction subset. The CLE/SSA achieves higher accuracy than ODE/SSA because it still could capture for the fluctuations in the fast reactions.

- The τ-leaping/SSA hybrid. The τ-leaping/SSA hybrid places in the middle between deterministic and stochastic hybrid. It is named as the *maximal timestep* algorithm in [126]. The key idea of τ-leaping/SSA hybrid is that the τ-leaping is applied to simulate the fast reactions while the slow reactions is simulated by an SSA variant. This hybrid technique bridges the gaps in the ODE/CLE integration and discrete event simulation described above. However, by applying a variant of τ-leaping for the fast subset makes this hybrid approach become more difficult to analyze the time-varying nature of slow reaction propensities. Thus, this technique puts an assumption that the changes in slow reaction propensities during a leap is less significant, and is ignored. This, of course, introduces an additional source of error to the simulation.

2.3.4 Stiff system simulation

The stiffness arises in systems consisting both fast and slow reactions where the fast reactions approach the stable state very fast. After rapidly transient time with a very short fluctuation due to fast reactions, the system becomes stable. The slow reactions then determine the system dynamics. The presence of multiple time scales in such system slows down the stochastic simulation significantly. In fact, SSA spends most of its simulation time for simulating fast reaction events; however, this is not corresponding to the system dynamics.

Many methods have been proposed for efficiently simulating the stiff systems. They are often based on two main techniques: the quasi-steady state assumption (QSSA) and the partial equilibrium assumption (PEA), which are used in the deterministic context and adapted to the stochastic simulation. The QSSA improves the simulation performance by removing intermediate and highly reactive species from the model, while PEA enhances the simulation by assuming fast reactions reaching equilibrium will remain always in that equilibrium state. The difference between QSSA and PEA is the object they focus on. The former focuses on the state, while the latter concentrates on the reactions. In the following we briefly review these techniques.

The QSSA-based stochastic simulation. In [129, 161], the QSSA stochastic kinetics is introduced to deal with stiffness. The system state x is divided into the set of primary species y and intermediate species z so that $x = (y, z)$. The intermediate species are assumed to be transitory and highly reactive. In other words, two following assumptions are made. First, the probability distribution of intermediate species z conditional on y approximatively satisfies the definition of the CME. That is $P(z|y,t)$ follows the form of chemical master equation in eq. 2.3. Second, the net rate of change for the conditional probability distribution of these intermediate species is approximatively equal to zero. It is equivalent that $dP(z|y,t)/dt \approx 0$. By these two assumptions, the stationary probability distribution of intermediate species $P(z|y)$ is more easier to derive. An analytic solution or a numerical computation can be conducted to sample the population of intermediate species. Having the knowledge of intermediate species, reaction propensities involving the primary species y for doing stochastic simulation become easier to derive. In fact, these propensities have the form $b_k(y) = \sum_z a_k(y,z)P(z|y)$.

Summing up, in each QSSA-based simulation loop two consecutive steps are done. First, the intermediate species z is sampled from the stationary distribution $P(z|y)$. They are substituted into the computation of propensities $b_k(y)$ involving primary species. And second, a SSA step is applied to find the next reaction firing based on propensities $b_k(y)$. Note that when a reaction firing only the population of primary species y is updated.

The PEA-based stochastic simulation. The slow-scale SSA (ssSSA) [23, 24] is an example of PEA. ssSSA proceeds as follows. It provisionally divides reactions into fast reactions, denoted R^f, and slow reactions, denoted R^s. The provisionally partitioning of reactions is decided only by their rate constants. The fast reactions are further assumed to remain always in equilibrium state upon reaching the equilibrium. Species whose population gets changed by a fast reaction are labeled as fast species S^f, the rest species is called slow species S^s. By this definition, a fast species clearly can change by a slow reaction, but the reverse direction is not true. The corresponding process $X(t)$ is thus divided into a fast process $X^f(t)$ and $X^s(t)$. Although the full state vector $X(t) = (X^f(t), X^s(t))$ obeys CME, each individual component is not. ssSSA overcomes this difficulty by introducing the definition of virtual fast process. More precisely, the virtual fast process $\tilde{X}^f$ contains the same species as the fast species $X^f(t)$ where all slow reactions turned off. Thus, the virtual fast process $\tilde{X}^f$ only depends on fast species, while the slow species are assumed constant. The $\tilde{P}(X^f, t)$ in this definition is completely described by CME.

The virtual process $\tilde{X}^f$, under the stiffness property, is assumed to be a stable process. It thus imposes two assumptions. First, the stationary distribution $\tilde{P}(x^f, \infty)$ exists. Second, the relaxation time of $\tilde{X}(t)$ to stationary asymptotic form, $\tilde{X}(t) \to \tilde{X}(\infty)$ happens very quickly (typically, smaller than the time to the next slow reaction event). With these two assumptions, the stationary distribution $\tilde{P}(x^f, \infty)$ is analytically solvable by e.g. a numerical method. Thus, the population of fast species involved in the virtual fast process can be computed without doing simulation. The simulation now only applies for slow reactions where the propensity of a slow reaction is adapted as follows. Let Δ_s be the time which is very large compared to relaxation time of $\tilde{X}^f(t)$, but also very small compared to the expected time to the next slow reaction. The probability one slow reaction R_j^s occurs in interval $[t, t + \Delta_s)$ is approximated by $a_j^s(x^f, x^s)\Delta_s$ where $a_j^s(x^f, x^s)$ is referred to as slow scaled propensity function of reaction R_j^s. It is given by:

$$a_j^s(x^f, x^s) = \sum_{x^{f'}} \tilde{P}(x^{f'}, \infty | x^f, x^s) \tag{2.14}$$

In conclusion, a ssSSA execution for sampling a trajectory is first numerically calculating the population of fast species. The fast species are indeed generated by randomly sampling the limited virtual fast process $\tilde{X}^f(\infty)$. Then, a SSA step is applied to select a next slow reaction to fire based on the slow scale propensities Eq. 2.14. Thus, in the manner, the simulation moves the system state forward in time by firing one slow reaction at a time with all fast reaction events ignoring.

2.3.5 SSA Extensions

Several extensions also have been introduced to cover different aspects of biochemical reactions systems by relaxing SSA underlying hypothesis. In this section we briefly review two such relaxations that are: the reaction with delays, and reaction with spatiality.

Reaction with delays. In SSA, the next reaction assumes to happen instantaneously. Biochemical reactions, in fact, will take a certain time to finish after they are initiated. The delayed time in biochemical reactions is thus inevitable, but it is often many orders smaller than the waiting time to the next reaction. The delayed time is therefore often ignored. The delayed time, however, will introduce a another source of noise and plays a crucial role in the development of the biochemical processes if it is in the order of the reaction time. For example, in [13], the effect of delayed time to the development of the gene expression has been observed. The system exhibits the stochasticity even the counterpart is deterministic. The delayed-time reactions could further use to reduce the deleterious effects of propagation noise.

Because of delays the Markovian property of SSA is invalidated. The instantly update of the system state caused by the reaction firing would end with an incorrect result. SSA thus should be modified to take into account the delayed time in reactions. In [19] an exact generalization of DM with delayed time reactions is introduced. The key steps of the algorithm are as follows. In each simulation step, the next reaction and its firing time is generated by DM. If the selected reaction is a delayed time reaction, the actual completed time of this reaction is stored in a stack. In the other case, the reaction is a non-delayed reaction. Its firing time is compared against with the time stored on the top of the stack. There are two cases. In the first case, the reaction time is less than the completed time of a delayed reaction, the system state will be normally updated by firing this non-delayed reaction. On the other hand, the selected reaction is discarded and the update of the delayed reaction is performed. An exact generalization of SSA covering all possible delays in reactions, called the delay stochastic simulation algorithm (DSSA), is introduced in [21]. An efficient modified of NRM for the delayed time reactions is also introduced recently in [6].

Reaction with spatiality. This extension considers relaxing the well-mixed assumption. The spatial homogeneous is easy to validate by *in vitro* experiments where the diffusion of molecular species is much faster than the reaction. However, it is in general not true for living cells. The species in cell environment is indeed very highly localized to improve cellular functions. The cell division, metabolic and signaling pathways, for example, strongly depend on the spatial information. The temporal evolution of SSA by reactions alone is not enough to reproduce important effects such as the molecular crowding, the excluded volume effect. It therefore should be extended to take into account the diffusion of species in space.

Several approaches adapted SSA to make it applicable for simulating the diffusion of species in space. The key idea of these extensions is diving the space into smaller subvolumes. The subvolume size length is chosen to be small enough so that the well-mixed assumption inside that subvolume is satisfied. SSA therefore can be applied to simulate reactions inside a subvolume. The diffusion of a species between subvolume is treated directly as a unimolecular reaction. The rate of diffusion is translated from the bulk diffusion by the Fick's law. The rate of the diffusion reaction, in general, should depend on the shape of subvolume. The modelling of the spatial information in such the way is often referred to as reaction-diffusion master equation (RDME). RDME is indeed a natural extension of CME for spatial heterogeneous.

The SSA-based stochastic simulation algorithms for a RDME are also introduced. A direct extension of DM to simulate RDME is proposed in [17, 151]. It uses a DM variant to select a reaction firing. If the selection is an biochemical reaction, the population of reactants involved in that reaction is updated. Then, only affected reactions in the current subvolume have to update propensities. In case the selection is a diffusion reaction, the diffusive species selects a random neighbor to move to and the population of this species in these subvolumes is updated. For anisotropy diffusion, the destination subvolume of a diffusive species should be explicitly defined instead of random selecting a neighbor. The affected reactions due to the diffusive species in both of these subvolumes update their propensities after that. The direct application of SSA for doing reaction-diffusion simulation, however, is often computation and/or memory inefficient. A possible improvement is dividing the selection of the reaction firing into two consecutive search steps. The first search discovers the subvolume containing the next reaction firing, then the second one retrieves out the next reaction firing within that subvolume. There are many possible combinations for doing these steps, e.g., two consecutive DMs. The next subvolume method (NSM) [45] is a notable formulation of spatial SSA extension in this way for sampling RDME. NSM is indeed a clever combination of NRM and DM. In NSM, the selection of the next subvolume using the idea of NRM. The putative times of subvolumes is precomputed and stored in an indexed priority queue. Since the smallest putative time is always on the top of the queue, the selection of the subvolume is in constant time. The next reaction firing in this subvolume will be found out by DM. After the next reaction firing is defined, the population of species and affected reaction propensities in subvolume(s) are updated depending on the type of the selected reaction. Then, the putative times of subvolume(s) are recalculated to reflect the changes. The priority heap of subvolume putative times is consolidated as well. By using the priority queue to select a subvolume, the time complexity of NSM is scaled logarithmic with the number of subvolumes.

The Gillespie Multi-particle Method (GMP) [137] is a different simulation approach to simulate the reaction-diffusion processes. It is different in sense that the reactions and diffusions are treated separately. The theory behinds GMP is known as the *operator splitting* technique [35, 36]. In essential, GMP pre-computes the diffusion time of a molecular species based on its diffusion constant and the subvolume size length. The time-axis is thus divided into small chunks of the diffusion times. During the simulation, if a diffusion event occurs, the corresponding species in a subvolume is distributed all over its neighbors. Between two diffusion events, reactions between species are simulated by DM. In [138] a hybrid τ-leaping (Hτ-leaping) algorithm is presented. It is working similar to GMP, but the diffusion time of all species is fixed instead. The Multinomial simulation algorithm (MSA) [100] also treats the reactions and diffusion separately but allows a molecule diffusing from a subvolume to any neighbors within a prescribed distance. It thus improves the spatial simulation if the number of diffusive events many orders larger than the reaction events. MSA uses a conditioned Multinomial distribution to approximate the number of molecules diffusing.

In literature, the particle-based spatial simulation algorithms have also been proposed. In these algorithms, the spatial information of each species is tracking directly. The diffusion of a species in space is explicitly model by a Brownian dynamic (BD). A reaction between molecules occurs if they are close enough. More specifically, if the distance between two molecules is less than the so-called *reaction radius*, the reaction could happen. The Smoldyn [7] is a direct application of the BD to simulate the reaction-diffusion at the particle level. However, the time step for moving a molecular species in space by a random walk has to choose small enough so that it does not miss reactions with other molecules. Green's Function Reaction Dynamics (GFRD) [163] solves this problem by a discrete event simulation. Thus the time step for doing a random work does not need to be fixed arbitrarily small. Recently, the combination of particle-tracking and RDME simulation are also proposed [55, 90].

Chapter 3

Tree-based search

3.1 Introduction

An insight understanding the mechanisms of regulatory effects in large cellular models gives many benefits, but also raises a great challenge for the implementation of the simulation algorithm. Both the search of the next reaction firing and the update affected reactions suffer the simulation performance. In this chapter, we focus our study to the impact of the search on the overall simulation performance, and contribute to its improvement by applying variants of a tree-based search. The update will be considered in the next chapter.

A linear search to determine the next reaction firing in DM, in principle, works with any biochemical reaction model. The accumulating sum of propensities a_j repeats until a reaction found. The time complexity of linear search, however, is increasing linearly with the number of reactions in the network, i.e., $O(m)$. Thus, except for some small models, the performance of linear search is often very slow.

A binary search is, of course, a more efficient method than a linear search (logarithmic vs. linear complexity). In order to exploit binary search the partial sums of propensities have to be precomputed and store in a tree structure. Hence, we will start by discussing the underlying data structures and algorithms used to apply binary search on complete trees with a dependency graph based update mechanism. Then, we study which tree structure will minimize the number of comparisons needed to find the next reaction firing.

We show, both in theory and in practice, that by using an underlying tree data structure to store reaction propensities the simulation time can be sensibly improved. Theory shows that our approach reduces search time from linear to logarithmic, although propensity updates now require logarithmic time instead of constant time. Theory also predicts the shape of the tree leading to optimal average search time. This turns out to be the Huffman tree [79], a device used in computer science for data compression. Experiments confirm that this tree indeed leads to faster simulation. We also study further the impact of tree-rebuilding approaches, by which the propensity Huffman tree is rebuilt when it becomes non-optimal caused of many reaction firings. Two strategies are proposed: the fixed time tree rebuilding and adaptive time tree rebuilding. The former strategy periodically rebuilds the tree after a fixed time, while the latter allows to rebuild the tree during the simulation depending on how the system evolves.

3.2 Complete Tree Search

A (binary) complete tree, is a binary tree completely filled at every level, except possibly the last; each node has exactly two children (internal node), or zero (leaf). For our purposes, leaves hold the reaction propensity a_j for $j = 1 \cdots m$, while internal nodes store the sums of values of their child nodes. Thus, at the top, the root holds the total sum a_0. Proposition 1 and the following discussion allow to store a complete tree on a contiguous array, hence improving cache-friendliness.

Proposition 1. *A complete binary tree with m leaves has exactly $2m - 1$ nodes.*

We therefore use an array with $2m - 1$ elements to represent a complete tree with m reaction propensities filled at the lowest level. In the array representation, a node at position i will have its two children at position $2i$ and $2i+1$. We then recursively from leaves to root construct the tree with the internal sums as in Algo. 2. Here, each element of the array TREE stores only the partial sums of the reaction propensities, so we simply need each cell to store a single value (a floating point double). In order to build up the tree, the number of reactions m must be an even number. In the case m is not one can add a dummy node (with propensity 0) as the last element of the array.

Algorithm 2 Building the complete tree

```
procedure: build_tree(position)
require: array TREE with $2m-1$ elements where elements from $m$ to $2m-1$ are filled with
    reaction propensities
1: if position is not leaf then
2:     build_tree(2position)
3:     build_tree(2position + 1)
4:     TREE[position] = TREE[2position] + TREE[2position + 1]
5: end if
```

Once having built the tree, to search for the next reaction firing we proceed as follows. Let r be a random number in $U(0,1)$, and ra_0 be the value we are looking for. Starting from the root, we travel down the tree, following the left or right branches according to whether the propensity sum stored in the left one is smaller than the search value. Whenever we take a right branch, we adjust the search value by subtracting it from the value stored in the parent. The whole procedure is outlined in Algo. 3. The procedure is correct, in the sense it finds the same leaf R_j as in Equ. (2.7), so each reaction indeed is chosen with the correctly desired probability a_j/a_0.

The reaction firing causes the system state change; therefore, we also have to update the propensity tree as well. For that, we use the dependency graph to keep the local affection between reactions and exploit the fact that the parent of node i is located at position $\lfloor i/2 \rfloor$. Hence, we only update the reactions affected and their ancestor nodes in the tree following the path from leaf to root. Since

Algorithm 3 Finding the next reaction firing

```
procedure: search(position, s)
require: properly set up array TREE, search value s
1: if position is leaf then
2:     return position
3: else if TREE[2position] $\geq$ s then
4:     search(2position, s)
5: else
6:     v = TREE[position] - s
7:     search(2position + 1, v)
8: end if
```

the average path length is $\log(m)$, the total cost for the simulation is stable $O(\log(m))$.

A particular order of reactions in the leaves of a tree in an implementation has impact on the update of the affected reactions. To illustrate the idea, we imagine a binary tree with two reactions which affect each other. The number of computation could reduce to a half if they are staying near each other, i.e., when they share the same parent comparing with the case they are put in different branches, i.e., they have different ancestors. In general reactions should be placed together so that they form a clique.

3.3 Huffman Tree Search

While storing reactions in a complete tree minimizes the *height* of the tree, corresponding to the average computation to search the next reaction firing, this does not lead to an optimal average-case performance. Indeed, consider the average number of comparisons performed during the search of the next reaction firing and denote this value by $T_m(C)$, we have:

$$T_m(C) = \sum_{j=1}^{m} w_j D_j \tag{3.1}$$

where m is the total number of reactions, D_j and w_j, respectively, are the depth of the leaf R_j in the tree and the weight corresponding to the probability the reaction R_j is being selected to fire. The reaction depth D_j is indeed the search length of firing reaction R_j. The SSA, by our formulation, therefore now has changed to find a representation to optimize $T_m(C)$.

In complete tree setting, the depths D_j are roughly equal, since all the leaves are in the last level or in the next-to-last one. So, we are performing the same number of computations in every cases i.e., the likely event of picking a fast reaction requires the same computational effort of the unlikely event of picking a slow reaction. It is simple to check that this choice leads to a non optimal $T_m(C)$. Consider the extreme case in which reaction 1 has 91% probability, while reactions $2, 3, 4$ have 3% probability each. In a complete tree, we would have $D_j = 2$, hence $T_4(C) = 2$. With a non-complete tree it would however be possible to move reaction 1 up in the tree ($D_1 = 1$), while moving the other reactions down ($D_j = 3, j > 1$). This leads to $T_4(C) = 1.18$ comparisons, which is better. Intuitively, we can improve the performance of the complete tree search, especially for multi-scale biochemical systems, which can be separated into fast and slow reactions. The main idea would then be to place fast reactions close to the root, while slow ones farther from it.

These facts are very closely related to well-known results in data compression. Indeed, the minimization of $T_m(C)$, which leads to optimal performance in our setting, is the purpose of the Huffman encoding for data compression. Huffman tree, in [79, 91], provides a possible construction to minimize $T_m(C)$. The fundamental idea there is to build the tree by repeatedly merging trees in a forest, which initially contains only trees with one node. At each step, the two trees whose roots (p and q) have the *smallest* weights (w_p and w_q) are merged. A new root pq is created and the two previous trees become the subtrees of pq. The pq node is assigned weight $w_{pq} = w_p + w_q$. This is repeated until the forest contains only one tree. From this, it is clear that in the final tree we have

$D_{pq}+1 = D_q = D_p$, where p, q, pq are the nodes involved in any merge. Hence, we obtain for any such p, q, pq:

$$\begin{aligned} T_m(C) &= \sum_{\substack{j=1\\ j\neq p,q}}^{m} w_j D_j + w_p D_p + w_q D_q \\ &= \big(\sum_{\substack{j=1\\ j\neq p,q}}^{m} w_j D_j + w_{pq} D_{pq} \big) + w_{pq} \\ &= T_{m-1}(C) + w_{pq} \end{aligned} \tag{3.2}$$

which relates $T_m(C)$ with $T_{m-1}(C)$. The above allows us to recall the main result for Huffman trees.

Proposition 2. *The Huffman tree gives the minimum value of* $T_m(C)$

Proof. Proof By induction on m. **Base case**: easy to check for $m = 2$. **Inductive case**: by the inductive hypothesis, the Huffman tree for $m - 1$ gives the optimum value for $T_{m-1}(C)$. By contradiction, suppose the Huffman tree for m is not optimal. So there is some tree having total number of comparisons $T'_m(C)$ such that $T'_m(C) < T_m(C)$. W.l.o.g. the smallest weights must be placed at lowest level. Hence, let p and q are nodes with smallest weight and their parent labeled pq. Using (3.2), we have $T'_{m-1}(C) + w_{pq} < T_{m-1}(C) + w_{pq}$ then $T'_{m-1}(C) < T_{m-1}(C)$, contradicting the inductive hypothesis. □

Since each node in Huffman tree has two children, Proposition 1 still holds. We therefore still use an array with size $2m - 1$ for representing the Huffman tree. Note that, however, we do not need m to be even in this setting. The elements at position from $m - 1$ to $2m - 1$ are filled by reactions as leaves. But, unlike for complete trees, each element in the array must point to its left and right child. Building a Huffman tree is done by employing a heap to extract the nodes p, q with minimum weight at each step.

Algorithm 4 Building Huffman tree

```
procedure: build_huffman_tree
require: An array TREE with $2m - 1$ elements, where the elements from $m$ to $2m - 1$ are
    filled
1: build heap H with elements $(m, w_1)$,... $(2m - 1, w_m)$, ordered according to $w_j$
2: for $position = m - 1$ down to 1 do
3:     extract top element ($p$, $w_p$) from H
4:     extract top element ($q$, $w_q$) from H
5:     TREE[$position$].VALUE = TREE[$p$].VALUE + TREE[$q$].VALUE
6:     TREE[$position$].LEFT = $p$
7:     TREE[$position$].RIGHT = $q$
8:     insert($position$, $w_p + w_q$) into H
9: end for
```

The Huffman tree we built in the Algo. 4 is stored in an array in which each element contains the fields: VALUE, LEFT, RIGHT. The partial sum is now stored in the VALUE field. The index of left and right subtree is indicated by LEFT and RIGHT, respectively. The same binary search procedure in Algo. 3 is applied to search the Huffman tree for the next reaction, except that now LEFT and RIGHT fields are used to travel the tree, instead of the previous formulas which work only for complete trees.

The update stage in the simulation is to reflect the changes to the propensity of reactions affected. Each element of array TREE stores the location of its parent node by an additional field, called PARENT, which is set in the Huffman tree building procedure. The path from a leaf to its root is thus easily to restored. Accompanying with dependency graph, we traverse upward this path to update reactions affected. In the following, we discuss about the weight function in the implementation of Huffman tree and the tree rebuilding when the tree becomes non-optimal.

By applying the Huffman tree to find the next reaction firing, we want to reduce the number of comparisons of SSA. A native candidate for the weight function is the propensity function a_j since this choice leads to less time spent for finding the next reaction. However, during the execution of the simulation, reaction firings affect their dependent propensities, which also could change rapidly. This happens, for example, whenever a reaction has a very large rate constant but a small number of reactant molecules. Its propensity will significantly change by a very large amount. Updating the values stored in the tree therefore could make the tree no longer optimal i.e., no longer an Huffman tree. In this case, we face the choice of either proceeding with a non-optimal tree (which could still be near the optimum, though), or rebuilding the Huffman tree. Rebuilding the tree is rather expensive, so we need a trade-off.

Our idea is postponing the reconstruction of the tree while the change of the weight is less significant. We thus keep on using a non-optimal tree for some predefined (and tunable) number of SSA steps. The choice of this parameter, however, only affects the performance, while the results are still exact.

3.3.1 Fixed time tree rebuilding

An intuitive and easy implementation for the tree rebuilding discussed above is to use a fixed number k, and consolidate the tree structure only once every k steps. This amounts to assuming that the weights do not change significantly during k simulation steps, so we can postpone the rebuilding without a large impact on performance. To compensate, we slightly modify weights w_j to cope with propensities changing rapidly. More precisely, we assign a higher weight to those reactions which are more likely to change.

For reaction R_j, we consider two sets: conflicts(R_j) as the collection of reactions that affect and compete with R_j

$$\text{conflicts}(R_j) = \{R_i | R_j \in \text{affects}(R_i), \text{reactants}(R_i) \cap \text{reactants}(R_j) \neq \emptyset\} \tag{3.3}$$

and favors(R_j) is the collection of reactions that affect and favor R_j

$$\text{favors}(R_j) = \{R_i | R_j \in \text{affects}(R_i), \text{products}(R_i) \cap \text{reactants}(R_j) \neq \emptyset\} \tag{3.4}$$

Table 3.1: Models with number of reactions and species

Model	Species	Reactions
Oregonator	8	5
Circadian Cycle	9	11
HSR of E. Coli	28	61
MAP Kinase Cascade	106	296

respectively, where reactants(R_j) and products(R_j) are the set of species taking part in reaction R_j as reactants and products. In terms of the dependency graph $DG(V, E)$, we have the following relation: $|$conflicts(R_j)$|$ + $|$ favors(R_j)$|$ = in-degree(R_j).

Then, we will estimate the probability a particular reaction occurring will increase (resp. decrease) the propensity of reaction R_j as $|$conflicts(R_j)$|/m$ (resp. $|$favors(R_j)$|/m$). For k simulation steps, the estimated weight of reaction R_j is:

$$w_j(a_j, k) = a_j + \alpha_1 k \frac{|\text{favors}(R_j)|}{m} + \alpha_2 k \frac{|\text{conflicts}(R_j)|}{m} \tag{3.5}$$

where α_1, α_2 are parameters denoting the average change amount. For simplicity, we assign these to the stochastic rate constant for the reaction at hand i.e, $\alpha_1 = -\alpha_2 = k_j$.

We evaluate and compare the performance of four algorithms: DM, NRM, Complete Tree Search and Huffman Tree Search. The simulation is performed on different models varying in size. Table 3.1 provides a summary of the number of reactions and species in each simulated systems. Before going to the details of the results, we give a brief description of these models.

The first two models we studied are the Oregonator and Circadian Cycle model. The underlying mechanism of the Oregonator dynamics contains both an autocatalytic step and a delayed negative feedback loop. It is a kind of chemical reaction that shows a periodic change in the concentrations of the products and reactants [5] where reactions and species involved are shown in Fig. 2.2. The second model is the simplified circadian cycle model in [166]. The circadian rhythm is a daily cycle in the biochemical processes of many living beings. The key mechanism of the circadian rhythm is the intracellular transcription regulation of two genes that is an activator and a repressor. Activator acts as the positive element in transcription in binding to promoter, while repressor acts as the negative element by repressing the activator.

The third model that we simulated is the heat shock response (HSR) process which occurs when cells are shifted to high temperature. The synthesis of a small number of proteins, called the heat shock proteins (HSPs), is rapidly induced. In E. coli, the response is controlled by the so-called σ-factor which is capable of binding to various regions of the DNA that stimulate the transcription of the particular gene under their control. When E. coli senses the raised temperature the special heat shock σ-factor called $\sigma 32$ will replace $\sigma 70$, which is the bound σ unit of RNA Polymerase (RNAP), to accelerate HSPs synthesis (see more details in [96]).

The last model is the mitogen-activated protein (MAP) kinase (MAPK) cascade. The MAP kinase signaling pathway is a chain of proteins in the cell that cascade a signal from a receptor on the surface of the cell to its nucleus. The signal begins when mitogens or growth factors bind to the receptor on the cell surface and ends when the cell responds a response pattern e.g., growth, differentiation, inflammation and apoptosis. The cascade is well-conserved which means this process can be found in a large number of cell types. The basic mechanism of this pathway is driven by three protein kinases: MAPKKK (such as RAS/Raf), MAPKK (such as MEK) and MAPK. The external stimuli activate the first element of the pathway, the MAPKKK. The activated MAPKKK phosphorylates MAPKK at two sites. The phosphorylated MAPKK then activates the MAPK through the phosphorylation on its threonine or tyrosine of the protein structure. MAPK can then act as a kinase for transcription factors, but may also have a feedback effect on the activity of kinases like the MAPKKK further upstream [93].

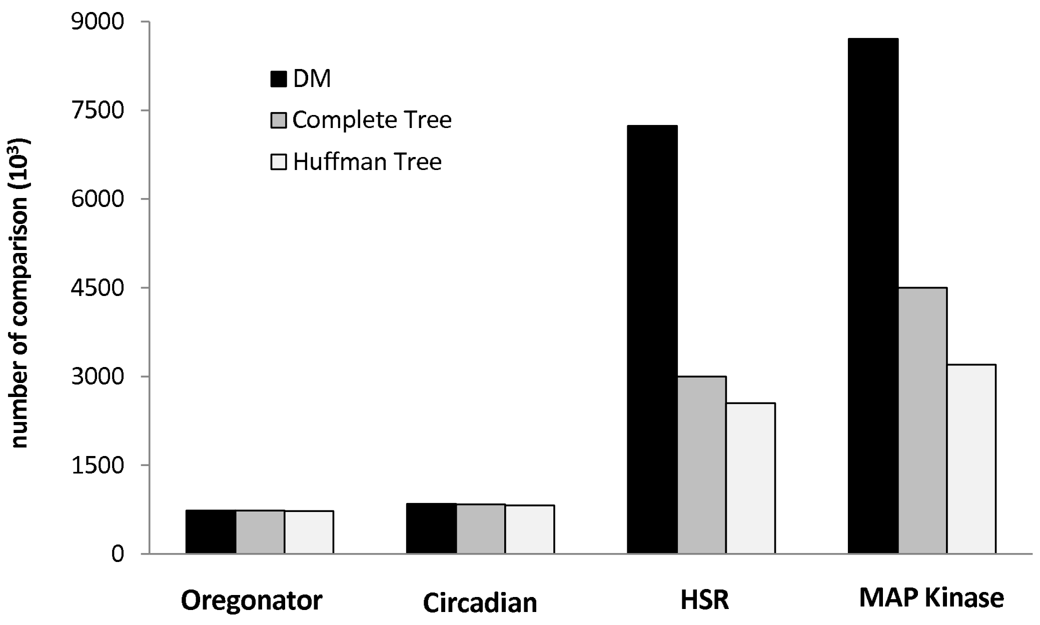


Figure 3.1: Number of comparisons performed by each algorithm. For Huffman Tree, we rebuild the tree every $k = 100,000$ steps.

The performance of four algorithms is reported in Fig. 3.1-3.2. The results have been computed for $500,000$ simulation steps on an Intel Core i5-540M processor. For the Huffman Tree Search, we had to pick a number k of steps after which we reconstruct the Huffman tree. In this experiment we chose $k = 100,000$, hence causing the tree to be rebuilt 5 times in the whole simulation.

In Fig. 3.1, we show the number of comparisons performed for finding the next reaction firing in each case. The NRM algorithm is not shown because the smallest putative time is always on the top of the priority queue used in NRM. In all the cases, the Huffman tree search performed the least number of comparisons. In simulating small models, the difference between linear search and binary search is not very significant. However, with the larger models binary search is nearly 50% faster than linear search, and Huffman Tree Search further improves on that by performing $\sim 20\%$ fewer comparisons than Complete Tree Search.

As shown in Fig. 3.2, simulating small models is not significantly affected

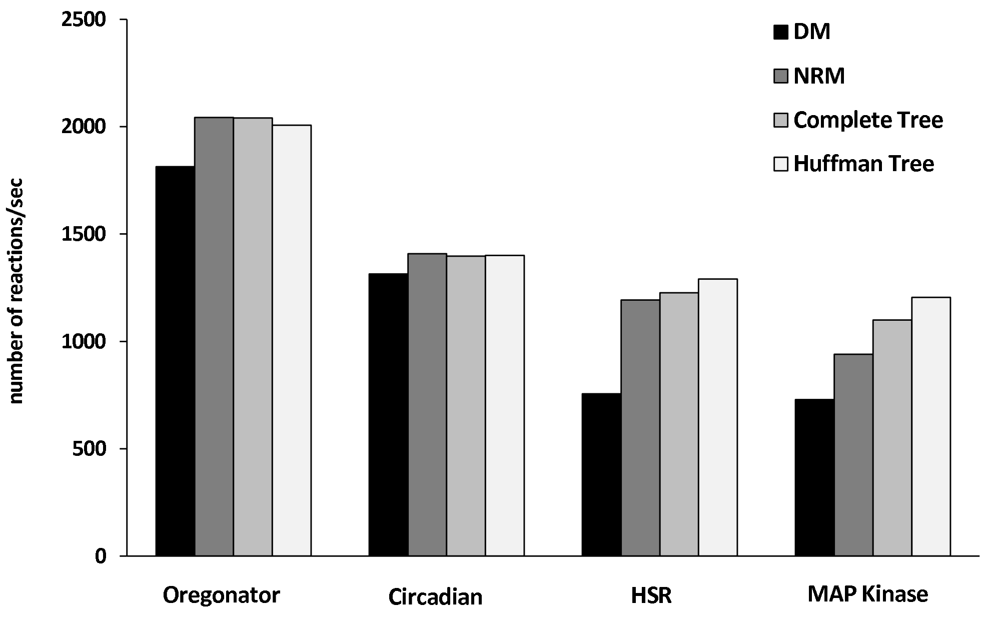


Figure 3.2: Overall performance in terms of reactions fired per second.

by the choice of the algorithm. This is intuitive, since in these models there is little room to improve both in search time and in update time, which contribute roughly in the same way to the overall performance. However, when the system is large, then search time dominates update time. In this case, search time significantly benefits from using an algorithm such as Huffman tree search, as our results for the MAP Kinase model show.

Picking an appropriate k to gain the best performance strongly depends on the problems at hand. More specifically, it depends on the changes of propensity function. In general, we could pick a large value k for systems which evolve near a stable state, so that changes in propensity are small. For unstable systems where the propensities sharply change frequently, by contrast, the value of k should be chosen small enough to capture such fluctuation. In practice, one can roughly estimate the value of k from a pilot simulation run, or move to an adaptive approach for tree rebuilding.

3.3.2 Adaptive time tree rebuilding

The tree rebuilding time k is model-dependent. Indeed, a periodic rebuilding tree with a fixed value k seems to be appropriate in simulating systems which are almost stable, so that propensity changes are small. In applying to arbitrary models, the performance is so very sensitive. We therefore improve the above approach by avoiding the use of a fixed number of steps, and instead use an adaptive approach in which we rebuild the tree only when there is a large change occurred i.e., when rebuilding seems to lead to a higher gain in performance.

Large changes of reaction propensity occur if the reaction rate constant is very large, hence even a small fluctuation in reactant population can lead to a very different reaction propensity. Large changes to propensity may also happen for reactions having a medium rate constant, but the population of reactants suddenly is increased by other fast reactions. These types of biochemical reactions are typically found in, e.g., *biochemical switches*. There, the system spends a bulk of time fluctuating near the stable state; however, when a random noise triggers the switch and this results in a dramatic change in the propensities of reactions (shown in Fig. 3.3). In that case, the tree should be rebuilt, or the search performance will suffer.

A straightforward procedure predicting such events is based on trial simulations, in which sample trajectories are collected. An analysis on the ensemble of trajectories is then performed to obtain roughly average times for the fluctuations in the system. These values therefore are used as the times to rebuild the Huffman tree. However, since the behavior of biochemical systems is inherently random, there will always be some difference between the predicted times and real simulation.

An intuitively less expensive approach is to dynamically check for changes in propensities caused by each simulation step. The advantage of this approach is that we detect the abrupt changes on the fly. To do so, we define an acceptance

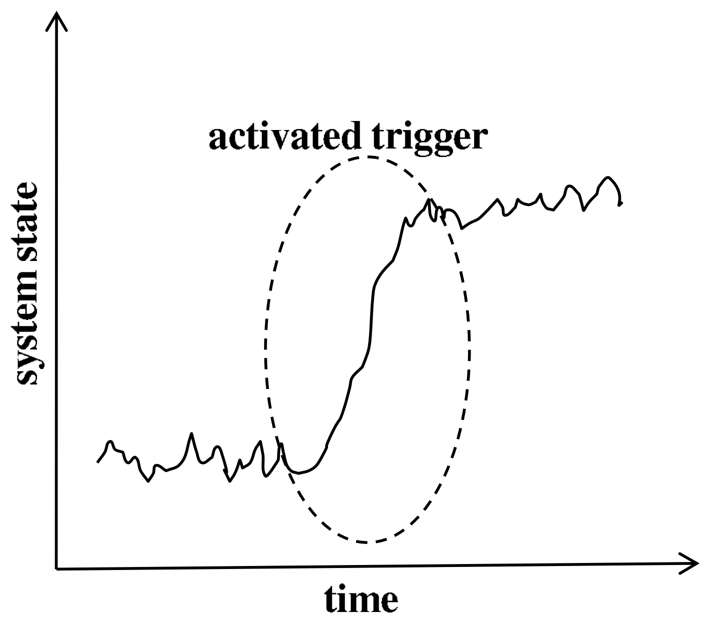


Figure 3.3: Random noise activates the trigger causing the system to abruptly exit of its stable state.

threshold δ, which is the largest change which does not trigger an immediately tree rebuilding. Let τ be the sojourn time until the next reaction fires. Then, the difference in propensity of an reaction R_j is computed as:

$$c_j(\tau) = a_j(x(t+\tau)) - a_j(x(t)) \tag{3.6}$$

where t is the current simulation time. When the above difference is high enough, i.e., $c_j(\tau) \geq \delta$, we then immediately restructure the Huffman tree.

In above we only consider abrupt single changes to propensities. We should also account for the fact that small updates, when applied many times, can also cause a significant change in propensities. To handle this case as well, we cumulatively sum all the propensity updates while simulating, as shown below.

$$s_j \mathrel{+}= \sum_{\tau} c_j(\tau) \tag{3.7}$$

Thus, we rebuild the entire tree when the cumulative sum is over the acceptance threshold i.e., when $s_j \geq \delta$.

Table 3.2: Gene expression model

DNA $\xrightarrow{k_1=5}$ DNA + mRNA
mRNA $\xrightarrow{k_2=5}$ mRNA + Protein
mRNA $\xrightarrow{k_3=1}$ $\emptyset$
Protein $\xrightarrow{k_4=1}$ $\emptyset$
DNA + IND_Protein $\xrightarrow{k_5=0.0001}$ DNA_INDProtein
DNA_INDProtein $\xrightarrow{k_6=100}$ DNA_INDProtein + mRNA

To compare the performance of the Huffman tree search with fixed and adaptive time rebuilding, we considered a gene regulation given in Table 3.2. There, a single gene is being translated into mRNA, which is then being transcribed into proteins. While there is no transcription factor binding to DNA, the transcription occurs at a medium rate. The system then slowly fluctuates for a long time. However, as soon as the transcription factor IND_Protein binds to DNA, it acts as an inducer, causing the transcription to happen at a larger rate by quickly producing a large amount of mRNA. In Fig. 3.4 we report the simulation runtime of these approaches. We measured the average time required to run a simulation for $500,000$ steps (disregarding the initial setup time for the algorithms).

In our gene expression model, the inducer protein IND_Protein has an important role while binding to DNA, since it accelerates the rate of mRNA production, which results in a large amount of proteins. This is because the last reaction in the model has very large rate $k_6 \gg k_1$, while its reactant population is small. The Huffman tree structure for applying binary search clearly should be consolidated when this reaction occurs.

The adaptive approach performs the reconfiguration at the correct time, by dynamically checking for propensity changes. Even if this requires a small overhead, still this leads to a better performance than those we obtained through the fixed approach, as Fig. 3.4 shows. By contrast, in the fixed approach, a small value of parameter k causes the tree to be rebuilt too many times. Here,

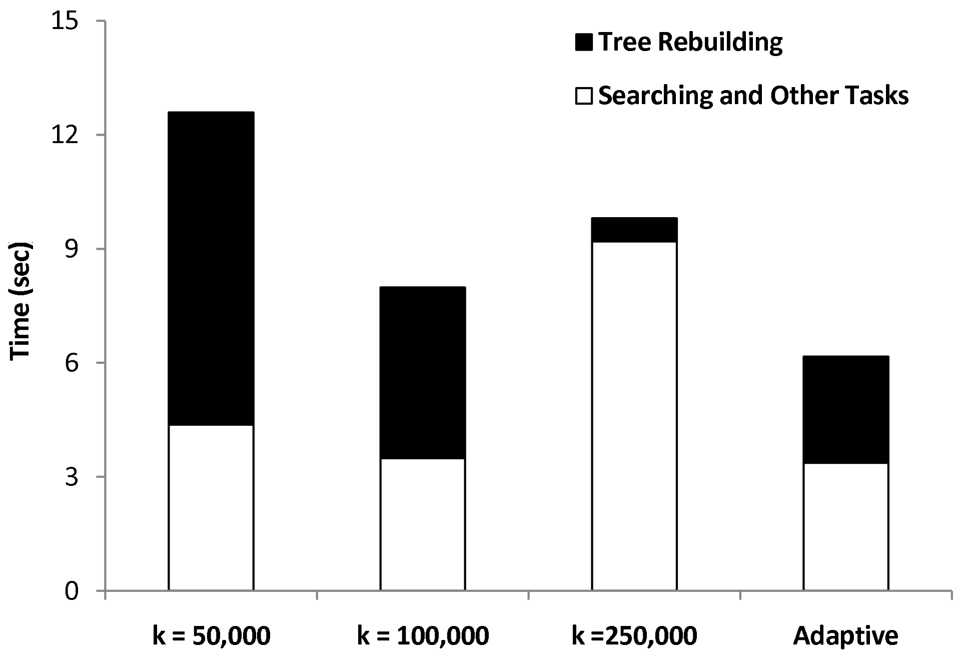


Figure 3.4: Simulation time of fixed time and adaptive time effort

although the tree is kept near the optimum, and less time is spent for searching, the rebuilding cost negates this advantage. On the other side, a higher value of k leaves the tree far from the optimum, causing search to be rather expensive and impacting one the overall performance.

3.4 Conclusion

In this chapter we apply binary search in trees to the SSA. We have shown the complexity of the search and update by a tree structure is reduced to logarithmic in the number of reactions. This feature makes it become more appealing to simulate large models. Further, we exploit the Huffman tree to reduce the number of comparisons needed for finding the next reaction firing. We proposed two strategies, the fixed time and adaptive time tree rebuilding, for keeping the tree close to the Huffman optimum during simulation, and studied their performance. The fixed time tree rebuilding is suitable for system near stable, while the adaptive tree rebuilding is more flexible.

Several improvements for the future are possible. For instance, in the studied approaches we either leave the Huffman tree as it is, or perform a complete rebuilding. One could then imagine to interleave full rebuilding, which is expensive, with a cheaper partial optimization. The latter would not restore the tree to an optimal case, but just improve it slightly. For instance, if a deep node in the tree is found to have higher propensity than a shallow node, we can quickly swap them and improve the tree. This optimization mechanism would be then similar to those used in garbage collection in computing systems, which is often split in frequent minor collections and rare major ones. As another approach, one could even explore the use of n-ary trees instead of binary trees.

Chapter 4

Rejection-based update

4.1 Introduction

Even though there are many efficient improvements of SSA introduced, most of them only focus on improving the search for the next reaction firing. For loosely coupled reactions in which the affected reactions have to update propensities roughly a small constant factor, the search largely affects to performance of the simulation. An efficient search procedure combined with a dependency graph update mechanism yields a great speedup gain; however, this is not always be the case for large cellular models. They are typically encompassed with a large number of interconnections and feedback loops. The dependency graph becomes very dense due to the highly coupled degree of reactions. Anytime the population of a species is changed, a large number of affected reactions have to recompute propensities. The costly propensity update is thus inevitable and contributes a significant portion to the simulation time. Hence, reducing the update time will substantially improve performance of the stochastic simulation.

Furthermore, in order to apply SSA, the reaction network has to be explicitly described in form of elementary reactions. In other words, the reaction network should not contain any abstraction. That is all intermediate products of all biochemical reactions have to be explicitly described. However, this emerges two difficult problems. First, if all possible combinations of molecular species are taken into account, the problem of state explosion can occur [120]. A signaling pathway, which enables the cell to sense the changes in its environment, is just an example. The signaling pathway is activated when a receptor is bound. The receptor often has many binding sites e.g., the phosphorylation site, the methylation site. A binding site changes the internal state of the receptor, and further controls and regulates the operations of the pathway. Due to the large number of possible binding combinations and their corresponding biological responses, a model with a quite limited number of species can derive a huge number of reactions (see e.g., [34] for details). Second, because of the incomplete knowledge in the full set of reactions, only the macroscopic behaviour of the biological system is observable. A typical example is the allosteric effect. This effect occurs when an effector molecule binds to the allosteric site of a targeted species e.g. a protein, an enzyme. The targeted species is then modulated and operated independently with the reactions in the system. This type of biological noise is referred to as the *extrinsic noise* [150]. These issues have augmented for the application of complex propensity functions in modelling biochemical reactions. The power law [39], for example, has been successfully applied to model such the cooperativity behaviour of biochemical reactions. However, evaluating a complex propensity function is indeed very time-consuming, and hence firmly increasing the computational burden in update the reaction propensities.

In this chapter, we are going to study the effect of propensity updates to performance of the stochastic simulation. We contribute to the improvement of SSA by introducing a new efficient algorithm, called Rejection-based SSA (RSSA). RSSA is an exact simulation algorithm, and is specifically tailored for the case in which propensity updates are time-consuming. It reduces the cost of propensities updates by avoiding and collapsing as much as possible the number of propensity updates. The propensity of a reaction is evaluated only as needed. In RSSA, the selection of a reaction firing is done through two steps. First, a candidate reaction is selected according to an over-approximation of its propensity. A rejection-based mechanism is then applied to recover the exactness of the algorithm. In the following, we are going to describe these steps in detail and devise improvements to this core of RSSA algorithm. Further, we discuss how to systematically optimize the tunable parameters of RSSA so to maximize its performance.

4.2 RSSA

RSSA improves over SSA by reducing the average number of propensity updates which have to be performed. Its key idea is to pre-compute an over-approximation of reaction propensities, and use that to select candidate reactions to be fired. Selected candidates are then inspected, and are either fired or (with low probability) rejected. The rejection mechanism is used to ensure that reactions are fired following exactly the distribution provided by SSA. RSSA takes advantage because it evaluates propensities infrequently (only as needed) by postponing and collapsing as much as possible their updates. When firing a reaction, with high probability, we do not need to recompute the propensity of all the dependent reactions. Hence, we avoid costly updates.

In the following, we first detail how RSSA samples a reaction firing with its firing time. We examine several possible choices with their implementations, and discuss the effects of such choices for different network sizes. Second, we provide a formal proof for correctness of RSSA. RSSA is exact in the sense it produces the same stochastic behaviour as SSA. Then, we focus on controlling the acceptance probability of a candidate reaction by varying the amount of over-approximation, hence indirectly controlling simulation performance. Several mechanisms are proposed which can be run at different levels, in a static or dynamic way. This allows RSSA to automatically adjust the acceptance probability depending on the current system state, so to adaptively optimize itself.

4.2.1 Selection of reaction firing

The over-approximation of reaction propensities is derived by giving a bound on the population of molecular species involved in the reaction. So, let $\underline{X}_i$ and $\overline{X}_i$, respectively, be an arbitrary lower bound and upper bound for each species S_i around the current population X_i. The interval $[\underline{X}, \overline{X}]$ is called the *fluctuation interval*. The current state vector X satisfies $\underline{X} \leq X \leq \overline{X}$ on each species. Then, we compute a lower bound $\underline{a}_j$ and an upper bound $\overline{a}_j$ for the reaction propensities a_j. Since a_j is a function f of the state vector X, the lower/upper bounds are computed by minimizing/maximizing such function over the whole fluctuation interval. Often, this function is monotonic, in that it increases whenever the species population increases. This is the case, e.g., for the mass action kinetics. If the monotonicity holds, one can simply let $\underline{a}_j = f(\underline{X})$ and $\overline{a}_j = f(\overline{X})$. In the case a complex propensity function f is used, one can e.g., apply numerical optimization techniques, or interval analysis [114] to recover the bounds. Note that we do not actually need the exact minimum and maximum: any (possibly tight) lower/upper bounds suffice.

Given propensity upper bounds and lower bounds, the selection of a reaction firing is composed of two steps as following. First, a candidate reaction R_j will be chosen with the probability $\overline{a_j}/\overline{a_0}$ in which $\overline{a_0}$ is the total sum of the upper-bound propensities i.e., $\overline{a_0} = \sum_{j=1}^{m} \overline{a_j}$. After the selection R_j is subject to a rejection test for validation. If R_j is accepted, it is fired and X is updated. Otherwise, R_j is discarded and we randomly select a new candidate.

The selection of the candidate is made randomly, assigning to each R_j the probability $\overline{a_j}/\overline{a_0}$. For this we need to apply a search algorithm for general discrete distributions. The general interface of the search is it takes a random number $r_1 \sim U(0, 1)$ as a parameter and returns a candidate reaction R_j with the corresponding probability $\overline{a_j}/\overline{a_0}$. Here, we can choose among several different algorithms. These algorithms are different in the speed and simplicity. A search with very fast marginal speed, however, requires to build complex underlying data structures, e.g., trees, hash tables before it could run. We discuss here three options for implementing of the search algorithm. We briefly discuss their time complexities as they are running. An experimental study is presented in the next section.

Linear search. The simplest search is the linear search. In linear search, the partial sum of upper-bound propensities is continuously accumulated, and the candidate reaction R_j is selected which is the smallest reaction index j satisfying the inequality $\sum_{k=1}^{j} \overline{a_k} > r_1 \cdot \overline{a_0}$. The advantage of linear search is that it does not require any complex data structures. In implementation we need an array size m to store m upper-bound propensities $\overline{a_j}$ for all $j = 1 \ldots m$. However, the time complexity of the search is linearly with the number of reactions i.e., $O(m)$. Hence, it often runs very slow with large models. Although the search can be improved if upper-bound propensities are sorted in the decreasing order, the complexity does not change in the worst case.

Binary search. A binary search can apply to find the next candidate reaction. The details of binary search is discussed in chapter 3. Essentially, a tree structure, e.g., a complete tree is needed to build before the search can be conducted. At the lowest level of the tree the upper-bound of reaction propensities are stored. The partial sums of propensity upper-bounds are stored in middle levels. The total sum $\overline{a_0}$ therefore will be stored at the tree root. When searching, only one root-to-leaf path of the tree is visited to find the candidate reaction R_j. The next branch is selected depending on the search value i.e., $r_1 \cdot \overline{a_0}$, with the partial sum stored in current internal node. The left branch is selected if the search value is less than the value stored in the internal node. The right branch is chosen otherwise. In case the right branch selected, the search value is adjusted by subtracting it with the value stored in this internal node. The search travels down with a new branch selection until a leaf is reached. Since the search complexity is linked to the length of the tree path, we could use a special tree structure i.e., the Huffman tree, to optimize the average search performance. In an implementation we can still use an array to represent the tree structure; however, compared with linear search the array has more elements. This is because we have to store also partial sums of propensity upper-bounds for the internal nodes, as well as the pointers to the children nodes. The time complexity for the (complete) binary tree search is logarithmic both in search and update the tree. This property makes it suitable for large models.

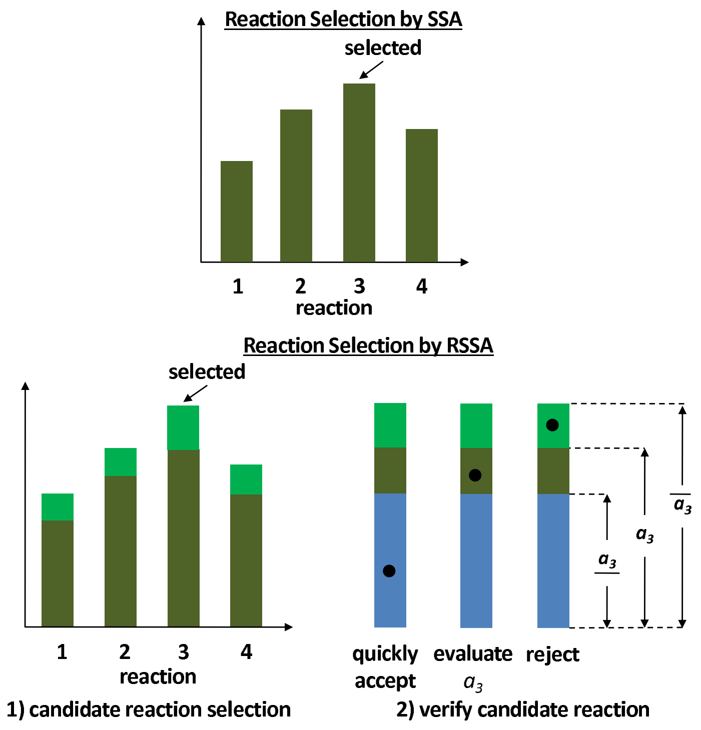
Lookup table search. A lookup table search is a very fast procedure to find a candidate reaction as comparing with a comparison-based search, e.g., linear search, binary search. The downside for using this search procedure is the pre-processing time which is needed to build the lookup tables. We have implemented and experimented with a well-known lookup search called the Alias method [43, 76, 169]. The theoretical foundation underlying the Alias method is a theorem stating that any discrete probability distribution over m values can be expressed as an equi-probable mixture of m two-point distribution. The probability vector is used in RSSA is the m-vector of probability $\overline{a_j}/\overline{a_0}$s. The set-up of the Alias method requires two tables each size m: a table, called cut-off table F, storing the probability of the first two-point distribution, and a second table, called alias table L, contains the alias of the second of the two-point distribution [94]. The pre-processing time to build these tables is linear with the number of values m [167]. The Alias method proceeds to search for a candidate reaction as follows. A random number $r_1 \sim U(0, 1)$ is first used to lookup the position of the equi-probable mixture. It is then rescaled to select which part in the two-point distribution. More specifically, the position $p = \lfloor m \cdot r_1 \rfloor$ of the two-point mixture is located. The first value in this two-point distribution stored in cut-off table F is loaded. It is compared against with $(m \cdot r_1 - p)$ to select the candidate reaction. If $((m \cdot r_1 - p) < F[p])$ the candidate reaction index $j = p$ will be returned. Otherwise, the candidate reaction index is the alias $j = L[p]$. The Alias method therefore requires only one comparison and at most two table accesses to select a candidate reaction.

When the candidate reaction R_j is defined, an acceptance-rejection procedure is applied to verify whether accept it to fire. The decision is made by its exact propensity a_j. For this validation step, we toss a (biased) coin with success probability $a_j/\overline{a}_j$. If the toss succeeds, we accept the candidate R_j, otherwise we reject it. The efficient simulation of this coin toss, however, is tricky since we do not know the exact value of a_j in advance, and we want to avoid computing it if possible. To achieve that, we extract a uniform random number $r_2 \leftarrow \mathcal{U}(0,1)$. We then check whether $r_2 \leq \underline{a}_j/\overline{a}_j$, which does not require us to compute a_j. If the check succeeds, then we know that $r_2 \leq \underline{a}_j/\overline{a}_j \leq a_j/\overline{a}_j$, hence we can accept R_j. Only when this test fails we indeed compute a_j, and then test r_2 against $a_j/\overline{a}_j$. Note that the computation of a_j is infrequently performed when $\underline{a}_j/\overline{a}_j$ is close to 1.

The clarification in reaction firing selection between SSA and RSSA is depicted in Fig. 4.1. The selection of a reaction firing in SSA is done in one step only by the exact reaction propensities, while RSSA instead uses the propensity upper bounds with two steps. Showing in the figure, reaction R_3 selected by SSA will be fired immediately after selected. In RSSA, candidate reaction R_3 has to be verified before it can be fired. This candidate reaction can even be rejected if the random value (the black dot in the figure) is larger than its exact propensity.

The Fig. 4.2 graphically demonstrates the improvement of RSSA over SSA in which reaction rate is modelled by the Michaelis-Menten expression. The figure shows the behaviour of RSSA on different regions of the Michaelis-Menten curve. From Fig. 4.2 we see that the fluctuation interval of the species can be widened without too much approximation of reaction propensity, and thus achieve a huge advantage in performance. More specifically, when the species population increases, it can be bound by a larger fluctuation interval, while the propensity range still gets narrower, allowing RSSA to rarely discard the picked

Figure 4.1: Select a reaction firing by SSA and RSSA

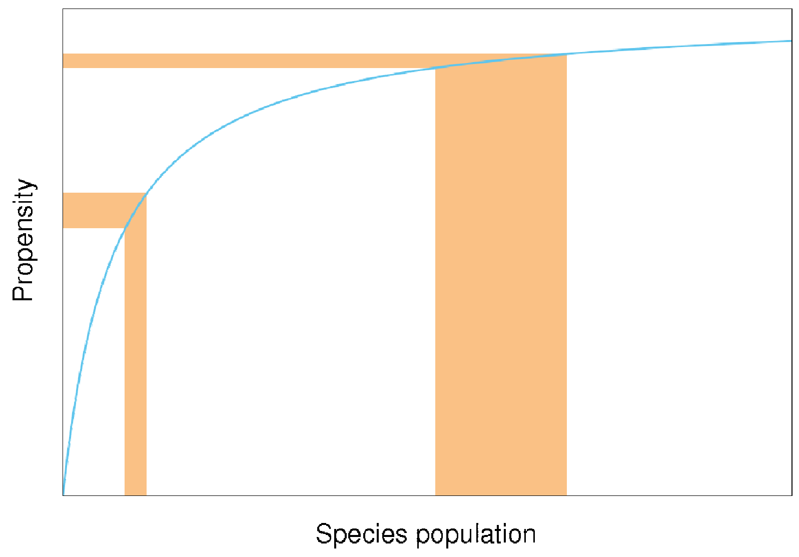


reaction.

4.2.2 Reaction firing time

By introducing propensity upper-bound $\overline{a_j}$ to select a reaction firing, rather than exact a_j, we add a probability the system will do a self-jump to its current state. We imagine that we have built a new transition rule for the candidate reaction R_j given the current state X at time t (see Fig. 4.3). There are two options for this candidate reaction: 1) moving to new state $X(t+\tau) = X(t) + v_j$ with rate a_j (w.r.t. the candidate reaction is accepted), or 2) still remaining in its current state $X(t+\tau) = X(t)$ with rate $(\overline{a_j} - a_j)$ (w.r.t. candidate reaction is rejected) in which τ is the waiting time. The total rate for m candidate reactions is $\sum_{j=1}^{m} [a_j + (\overline{a_j} - a_j)] = \overline{a_0}$. Therefore, the waiting time for all these m transitions is exponential distributed with mean $1/\overline{a_0}$. The firing time of an

Figure 4.2: The behaviour of RSSA on different regions of the Michaelis-Menten curve



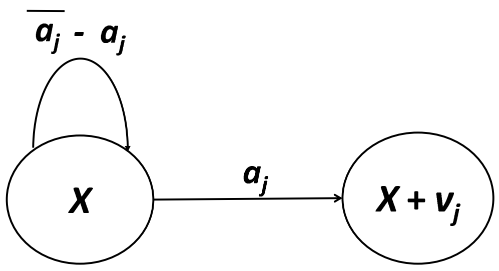
accepted reaction is the accumulated times of the consecutive rejected candidate reactions. In probability theory, the firing time of the accepted reaction is an $Erlang(k, \lambda)$ distribution with rate parameter $\lambda = \overline{a_0}$ and shape parameter k is the number of trials until that reaction is accepted.

We use the convolution method to sample the $Erlang$ distribution. Hence, we count the number of trials k until a new state transition occurring due to a reaction accepted. Then, k uniform random numbers, denoted by $u_i \sim U(0, 1)$ for $i = 1 \ldots k$, are generated. The firing time is computed by:

$$\tau = (-1/\overline{a_0}) \ln(\prod_{i=1}^{k} u_i) \tag{4.1}$$

In practice, we can approximate the reaction firing time τ by the mean of the corresponding $Erlang$ distribution i.e., $k/\overline{a_0}$. If the number of trials k is large, a lot of random numbers used in generating the $Erlang$ distribution will be saved.

Figure 4.3: Transition rule for a candidate reaction



4.2.3 The RSSA algorithm

Following the discussions in previous section, we now present the overall of the RSSA algorithm. The outline is given in the Algo. 5. The simulation repeats (by the **while** loop at line 1) until the current time t passes over a predetermined simulation time T_{max}. The code inside the simulation loop is logically divided into three parts: 1) preparing data structures for selecting the next reaction firing (line 2 - 4), 2) deciding which reaction fires next and its firing time (line 8- 21), and 3) updating and maintaining the system state due to a reaction firing (line 22 - 25).

The preparation starts at line 2. First, the fluctuation interval $[\underline{X}, \overline{X}]$ of the current system state X is defined. Given the fluctuation interval, we will compute the upper-bound propensity $\overline{a_j}$ and lower-bound propensity $\underline{a_j}$ of a particular reaction R_j. The usage of lower-bound propensity will speed up the acceptance process when the evaluation of the propensity is time-consuming. The corresponding total upper-bound propensity $\overline{a_0}$ sums up all $\overline{a_j}$.

The selection of the next reaction firing is done through a loop from line 8 - 20. The loop repeats until the flag $accepted$ is set to **true**. In each iteration three

Algorithm 5 RSSA procedure

```
1:  while $t < T_{max}$ do
2:      define the fluctuation interval $[\underline{X}, \overline{X}]$ of current state $X$
3:      compute the upper-bound propensity $\overline{a_j}$ and lower-bound propensity $\underline{a_j}$ for each reaction $R_j$
4:      compute the total upper-bound sum $\overline{a_0}$
5:      repeat
6:          set $u = 1$
7:          set $accepted$ = false
8:          repeat
9:              generate three random numbers $r_1, r_2, r_3$ from uniform distribution $U(0, 1)$
10:             search for a candidate reaction $R_j$ with probability $\overline{a_j}/\overline{a_0}$ by $r_1$
11:             if $r_2 \leq (\underline{a_j}/\overline{a_j})$ then
12:                 $accepted$ = true
13:             else
14:                 evaluate $a_j$ with current state $X$
15:                 if $r_2 \leq (a_j/\overline{a_j})$ then
16:                     $accepted$ = true
17:                 end if
18:             end if
19:             set $u = u \cdot r_3$
20:         until $accepted$
21:         set transition time $\tau = (-1/\overline{a_0}) \ln(u)$
22:         update time $t = t + \tau$
23:         update state $X = X + v_j$
24:         store/handle data
25:     until $X(t) \notin [\underline{X}, \overline{X}]$
26: end while
```

random numbers r_1, r_2, and $r_3 \sim U(0,1)$ are generated, respectively. The first two numbers is used to decide which reaction occurring, while the last random number r_3 is accumulated up to calculate the reaction firing time (by line 21).

At line 10, we use r_1 to randomly retrieve a candidate reaction R_j with the probability $\overline{a_j}/\overline{a_0}$. A particular search discussed in the previous section can be applied; however, it may require to set-up underlying data structures at the preparation step. Then, we decide whether to accept this candidate reaction firing or to reject it. At line 11, we compare $r_2 < \underline{a_j}/\overline{a_j}$. If this inequality is satisfied, we immediately accept the candidate reaction R_j firing without evaluating its exact propensity. We only compute the actual propensity in case this condition fails. For this situation, we evaluate the reaction propensity a_j (line 14). Then, if $r_2 < a_j/\overline{a_j}$ reaction R_j is accepted. We then move to calculate its firing time.

The reaction firing time τ is computed by line 21 i.e., $\tau = (-1/\overline{a_0})\ln(u)$, in which variable u is defined at line 6. It is, in fact, a implementation of the convolution method for the $Erlang$ distribution by Eq. 4.1 discussed in the previous section. RSSA multiplies variable u in every validation step by a uniform random quantity r_3 at line 19.

Consequently, from line 22 - 24, we finish a simulation step. The system moves to new state $x + v_j$ caused by reaction R_j firing. We advance the simulation clock to new time $t + \tau$. The current simulation data could be stored to external storage for further processing.

When we update the state (line 23) we do not have to update propensities for dependent reactions as in SSA. This is especially beneficial when the reaction network comprises reactions having a large number of dependencies. In that case, SSA has to recompute the propensities for each of them, while RSSA simply skips this step. On the other hand, RSSA has to check whether the new state X still belongs to the fluctuation interval by line 25. This requires only a few comparisons, since only a few species were affected by the fired reaction.

If the system state is still confined in its fluctuation interval, the new reaction firing selection is executed. In the uncommon case in which the system state leaves the interval, i.e. $X \notin [\underline{X}, \overline{X}]$, we exit the loop in line 25 so that a new fluctuation interval is defined. At that time, new upper-bounds and the lower-bounds of reaction propensities as well as the supported data structures for the search have to be recomputed.

4.2.4 Proof of correctness

We now show the correctness of the RSSA algorithm. The correctness, in this sense, means RSSA selects the next reaction firing R_j with the same probability as SSA i.e., a reaction R_j is selected with corresponding probability a_j/a_0. This result is stated in Proposition 3. In other words, RSSA produces the same stochastic behavior as SSA.

Proposition 3. *RSSA is exactly choosing a reaction R_j to fire with probability a_j/a_0. In addition, its firing time is exponential distribution with rate a_0.*

Proof. At a specific time t with current system state $X \in (\underline{X}, \overline{X})$, let $Pr(R_j)$ be the probability a candidate reaction R_j is selected and accepted to fire. $Pr(R_j)$, by the chain rule, is the multiplication of two probabilities: the probability of R_j is selected as a candidate, and the probability it is accepted. Hence, it given by:

$$\begin{aligned}\Pr(R_j) &= \Big(\frac{\overline{a_j}}{\overline{a_0}}\Big) \cdot \Big(\frac{a_j}{\overline{a_j}}\Big) \\ &= \frac{a_j}{\overline{a_0}}\end{aligned} \tag{4.2}$$

Now, let $Pr(R)$ be the probability an arbitrary reaction which is selected and accepted with current system state. We have $Pr(R)$

$$\begin{aligned}\Pr(R) &= \frac{\sum_{j=1}^{m} a_j}{\overline{a_0}} \\ &= \frac{a_0}{\overline{a_0}} \qquad (4.3)\end{aligned}$$

Thus, using the conditional probability, we can derive the probability reaction R_j is selected and accepted given an arbitrary candidate reaction R is selected and accepted. That is:

$$\begin{aligned}\Pr(R_j|R) &= \Big(\frac{a_j}{\overline{a_0}}\Big)/\Big(\frac{a_0}{\overline{a_0}}\Big) \\ &= \frac{a_j}{a_0} \qquad (4.4)\end{aligned}$$

For the second statement, let f_τ be the PDF of the firing time of the accepted reaction R_j. We will prove that it has the exponential distribution with rate a_0 i.e., $f_\tau(x) = a_0 \cdot e^{-a_0 x}$. In following let suppose the number of trials before reaction R_j accepted is denoted by k. The following derivation makes use the fact that a reaction is accepted at trial k (i.e., $k-1$ trials previous are rejected) following a geometric distribution with success probability $a_0/\overline{a_0}$.

We have:

$$
\begin{aligned}
f_\tau(x) &= \frac{\partial}{\partial x}\,\mathbb{P}(\tau \le x) \\
&= \frac{\partial}{\partial x}\,\textstyle\sum_{k_0=1}^{\infty} \mathbb{P}(\tau \le x \mid k = k_0)\cdot \mathbb{P}(k = k_0) \\
&= \frac{\partial}{\partial x}\,\textstyle\sum_{k_0=1}^{\infty} F_{Erlang(k_0,\overline{a_0})}(x)\cdot \frac{a_0}{\overline{a_0}}\cdot (1-\frac{a_0}{\overline{a_0}})^{k_0-1} \\
&= \textstyle\sum_{k_0=1}^{\infty} \frac{\partial}{\partial x}\, F_{Erlang(k_0,\overline{a_0})}(x)\cdot \frac{a_0}{\overline{a_0}}\cdot (1-\frac{a_0}{\overline{a_0}})^{k_0-1} \\
&= \textstyle\sum_{k_0=1}^{\infty} f_{Erlang(k_0,\overline{a_0})}(x)\cdot \frac{a_0}{\overline{a_0}}\cdot (1-\frac{a_0}{\overline{a_0}})^{k_0-1} \\
&= \textstyle\sum_{k_0=1}^{\infty} \frac{\overline{a_0}^{k_0}\cdot x^{k_0-1}\cdot e^{-\overline{a_0}x}}{(k_0-1)!}\cdot \frac{a_0}{\overline{a_0}}\cdot (\frac{\overline{a_0}-a_0}{\overline{a_0}})^{k_0-1} \\
&= a_0\cdot e^{-\overline{a_0}x}\cdot \textstyle\sum_{k_0=1}^{\infty} \frac{(\overline{a_0}-a_0)^{k_0-1}\cdot x^{k_0-1}}{(k_0-1)!} \\
&= a_0\cdot e^{-\overline{a_0}x}\cdot e^{x\cdot(\overline{a_0}-a_0)} \\
&= a_0\cdot e^{-a_0 x} = f_{Exp(a_0)}(x)
\end{aligned}
$$

$\square$

From the proof, we derive that the acceptance probability of the acceptance-rejection step in selecting a reaction firing of RSSA is bound. In fact, let $Pr(acceptance)$ be the acceptance probability. We have that

$$\underline{a_0}/\overline{a_0} \le Pr(acceptance) = a_0/\overline{a_0} \le 1 \tag{4.5}$$

Because the lower-bound and upper-bound propensity are functions of the given fluctuation interval, we could adjust the acceptance probability to achieve a desired probability through tuning this interval.

4.2.5 Fluctuation interval control

Once a candidate reaction is selected, RSSA decides whether to accept or reject the candidate reaction. We can adjust the acceptance probability of a candidate reaction through controlling the fluctuation interval $[\underline{X}, \overline{X}]$. In general, the smaller the interval we use, the higher acceptance probability a candidate has, however propensity updates become more frequent. In the special case where the fluctuation interval $[\underline{X}, \overline{X}]$ degenerates into state X the acceptance probability is 100%. In other words, RSSA reduces to the original SSA. On the other side, if we increase the fluctuation interval, we reduce the number of propensity updates. We can even widen the flucutation interval so that no update occurs during the simulation; however, the candidate reaction will be rejected frequently. In that case, the acceptance probability is decreased significantly. The selection of the next reaction firing therefore has to be repeated frequently. Summing up, it is important in RSSA to control the fluctuation interval so that we can control the acceptance probability, and thus the simulation performance. The simulation performance is then optimized when the search and update costs are balanced.

Three mechanisms are discussed below. The simplest one is the uniform fluctuation rate in which all species use the same rate. It has both advantages and disadvantages. On the positive side, the calculation of fluctuation interval is fast, requiring only vector computation. However, it is quite not suitable for the models having species fluctuating in different scales. For example, consider the case in which some species are involved in fast reactions, and are modified frequently while other species fluctuate slower. The application of uniform rate in this case is clearly inefficient. Using a non-uniform rate or an adaptive rate appears to be better since they would allow to control the fluctuation interval of each single species. The latter approach is the most flexible. It allows to adjust the fluctuation rate at runtime to improve the acceptance probability depending on the phase of the system. The most advantageous application of adaptive rate control is on those models where the population of some species fluctuates from very high to very low and vice versa. It is clear that we should dynamically change the fluctuation rates of these species to optimize the acceptance probability. Also, an absolute interval size (instead of a $\%$) can be preferred in case the population of a species is very low (say e.g., less than 25) than the relative rate.

Uniform fluctuation rate. This is the simplest procedure to control the fluctuation interval. All molecular species are assigned with the same rate δ. Then, the population of a molecular species S_i is assigned to an fluctuation interval $[X_i(1-\delta), X_i(1+\delta)]$.

When a reaction is selected to fire, the populations of the molecular species involved in this reaction is updated. The new system state is checked to satisfy its fluctuation interval constraint i.e., $X \in [\underline{X}, \overline{X}]$. There are two possible outcomes. If the constraint is satisfied, that is the system state is still confined in the fluctuation interval, the simulation continues without doing any update to the underlying data structures. Otherwise, the system state X has moved out of its assigned interval. The new fluctuation interval has to be computed. It is given by (using vector notation) as $[X(1-\delta), X(1+\delta)]$. The new upper-bound of reaction propensities as well as the underlying data structures have to be re-computed. Then, the new simulation iteration is executed.

Non-uniform fluctuation rate. The main idea of non-uniform fluctuation rate is to assign different rates to molecular species. It provides flexibility to control in detail the fluctuation interval of each species. On the other hand, it also requires more computational effort. This approach is indeed a generalization of the uniform fluctuation rate. It is intuitively useful to apply to the systems where some molecular species fluctuate more frequently in a larger interval than other species.

Let us consider a multiscaled system, in which reactions can be separated into fast and slow reactions. A fast reaction is selected to fire most of the time during the simulation. Fast species involved in the fast reactions therefore change more frequently. It seems useful then to use larger fluctuation rates for fast species, and smaller ones for slow species.

Thus, the application of uniform fluctuation rate for these systems seems to be less efficient. We should assign a larger rate for fast species, hence decreasing the number of updates, and a smaller rate for slow species, hence improving their acceptance probability. In order to implement the non-uniform rate approach to multiscaled models, we first divide the reactions into fast reactions and slow reactions. It is possible to do that because the system satisfies the multiscaled condition. The classification of reactions depends on the reaction rate. The reactions having large reaction rate will be assigned to fast reaction group. The species which are involved in fast reactions are labeled fast species, while the rests are called slow species. A fluctuation rate δ_{fs} will be applied for the fast species while slow species are assigned a smaller rate δ_{ss}. Using the assigned rate, we can calculate the fluctuation interval of each species type.

Indeed, we can generalize the non-uniform rate approach used in multiscaled models so that each species is assigned a unique rate. That is done by letting δ_i be the fluctuation rate of species S_i. We then assign to that species the fluctuation interval $[X_i(1-\delta_i), X_i(1+\delta_i)]$. In implementation, a lookup table can be used to store the fluctuation rate of each species to speed up the retrieving.

Adaptive fluctuation control. The mechanisms discussed above are static fluctuation control systems in the sense they apply fixed fluctuation rates during the simulation. In some models, the population of some species can change significantly during the simulation, e.g., moving from very highly abundant to very low and vice versa. The rate for such molecular species should therefore be changed to adapt the fluctuation interval of the involving reactions. Indeed, RSSA allows to adjust the fluctuation interval adaptively during the simulation depending on the availability of species in the system (for instance, using a threshold). If the population of the species is high we assign it a larger rate. During the simulation, when it gets down to low abundance, we apply a smaller rate instead to improve the acceptance probability. In this way, simulation can achieve a better performance.

Let us consider the application of adaptive interval control, and demonstrate its efficiency to tackle the case in which the species population is at very low copy number. We combine the relative fluctuation rate and fixed interval size to overcome this problem. In order to exploit the adaptive interval control we first need to set a threshold value λ. Second, for each species S_i, we apply a fluctuation rate δ_i hence using the interval $[X_i(1-\delta_i), X_i(1+\delta_i)]$ whenever $X_i \geq \lambda$. Instead, if the population of S_i gets lower than threshold value i.e., $X_i < \lambda$, we will apply a fixed (absolute) fluctuation interval Δ. The population X_i of species S_i now fluctuates in the interval $[X_i - \Delta, X_i + \Delta]$.

Following this example, we can extend the idea of adaptive fluctuation control to models having many phases. The population of species in each phase will be assigned a specific rate. To do that we set a threshold λ_i^k corresponding to species S_i at phase k. During the simulation if the molecular species S_i is bound to this threshold, a specific fluctuation rate δ_i^k will be applied. This advantage allows an automatic adjustment of the fluctuation interval of species during the simulation depending on the phase of the system state.

4.3 Experimental results

In this section we experiment with the performance of RSSA using three biochemical reaction models: 1) Fully connected reaction model, 2) Multiscaled reaction model, and 3) Gene expression model. Table 4.1 summarizes the properties of simulated models. The first two models are artificial ones, crafted so

Table 4.1: Summary of models

Model	Species	Reactions
Fully connected reaction model	N	N(N-1)
Multiscaled reaction model	N + M	N(N-1) + M
Gene expression model	5	8

to study the performance of the RSSA in different settings. These are highly coupled reaction networks where one reaction firing causes a large number of affected reactions to update their propensities. Thus, most of the simulation time would be spent for propensity updates in standard SSA. The last model is a real-world model that we use to demonstrate the improvement of RSSA. We consider different types of chemical reaction kinetics (i.e., mass-action kinetics and Hill kinetics) applied to this last model. Even if this model is quite small (having just 8 reactions), the employed chemical kinetics are non-trivial, hence the propensity updates require a significant computational cost. By optimizing such updates, we aimed to observe a large effect on the performance of the simulation. In this way, we assess the RSSA efficiency over conventional stochastic simulation methods.

Three algorithms are tested including: Gillespie's Direct Method (DM), Next Reaction Method (NRM), and RSSA. In RSSA, we further consider three implementations for searching the candidate reaction: 1) Linear search (RSSA-Linear), 2) Binary search (RSSA-Binary) and 3) Alias lookup search (RSSA-Lookup). All these simulation algorithms are implemented in Java and run on Intel i5-540M processor. In each case, the simulation ran for $2 \cdot 10^6$ reaction steps. The simulation data are recorded for 10^5 steps. The experimental measures exclude the initialization time, hence focusing only on the main simulation loop of each method.

4.3.1 Fully connected reaction model

The fully connected reaction network is a highly coupled reaction model we used to benchmark the performance of the simulation algorithms. It consists of N chemical species denoted S_i which reversibly convert into each other species S_j at a reaction rate k_i. A general form of reaction in this model is:

$$R_i : S_i \xrightarrow{k_i} S_j, i \neq j = 1 \ldots N$$

In our experiment the initial population of each species is set to 100. The propensity function is derived following the usual mass-action kinetics.

In Table 4.2, we measure the performance of the algorithms when increasing the number of species N. In this table, the execution time is the total time (including both the search and update time) spent to run the simulation. The update time counts the time spent for recomputing the propensities upper-bounds and rebuilding the needed data structures when the system state leaves the given fluctuation interval. The uniform fluctuation rate mechanism was used to control the fluctuation interval in RSSA. Three different values of δ are considered: $10\%, 20\%$ and 30%, respectively.

In this fully connected model the number of affected reactions is linearly increasing with the number of species N. In fact, there are $N - 1$ affected reactions having to update their propensities each time a reaction fires. For this high coupled degree, the update time is shown to largely contribute to the simulation runtime as N is increased. For example, the update time of DM and NRM in case $N = 100$ contributes up to 93% and 99%, respectively, to the total simulation runtime. This results in a rather low performance of these algorithms. In contrast, RSSA efficiently controlled the update of propensity. Therefore, it has significantly reduced the simulation time. For example, with $\delta = 20\%$ RSSA-Linear is roughly 10 times faster than DM, NRM with the same configuration. In this network size $N = 100$, RSSA-Binary with uniform fluctuation rate $\delta = 30\%$ received the best performance (approxiamtely 65 times

Table 4.2: Performance of algorithms on fully connected reaction model

N	Algorithm		Execution Time (ms)	Update Time (ms)	Acceptance Prob. (%)
5	DM		4234	3245	
	NRM		4935	4272	
	RSSA-Linear	uniform rate ($\delta = 10\%$)	1719	100	91.25
		uniform rate ($\delta = 20\%$)	1727	10	83.59
		uniform rate ($\delta = 30\%$)	1808	3	77.23
	RSSA-Binary	uniform rate ($\delta = 10\%$)	1764	120	91.28
		uniform rate ($\delta = 20\%$)	1790	21	83.65
		uniform rate ($\delta = 30\%$)	1851	7	77.15
	RSSA-Lookup	uniform rate ($\delta = 10\%$)	1755	144	91.24
		uniform rate ($\delta = 20\%$)	1742	29	83.60
		uniform rate ($\delta = 30\%$)	1880	11	77.17
10	DM		8632	7561	
	NRM		9862	9066	
	RSSA-Linear	uniform rate ($\delta = 10\%$)	2182	307	91.29
		uniform rate ($\delta = 20\%$)	2032	67	83.64
		uniform rate ($\delta = 30\%$)	2111	21	77.14
	RSSA-Binary	uniform rate ($\delta = 10\%$)	2177	383	91.32
		uniform rate ($\delta = 20\%$)	2021	84	83.60
		uniform rate ($\delta = 30\%$)	1998	29	77.15
	RSSA-Lookup	uniform rate ($\delta = 10\%$)	2243	535	91.27
		uniform rate ($\delta = 20\%$)	1941	118	83.60
		uniform rate ($\delta = 30\%$)	1960	34	77.23
50	DM		78083	70941	
	NRM		80208	78753	
	RSSA-Linear	uniform rate ($\delta = 10\%$)	12389	3615	91.28
		uniform rate ($\delta = 20\%$)	10482	918	83.61
		uniform rate ($\delta = 30\%$)	10708	368	77.20
	RSSA-Binary	uniform rate ($\delta = 10\%$)	6490	4094	91.27
		uniform rate ($\delta = 20\%$)	3503	1002	83.62
		uniform rate ($\delta = 30\%$)	3169	416	77.18
	RSSA-Lookup	uniform rate ($\delta = 10\%$)	19222	16999	91.29
		uniform rate ($\delta = 20\%$)	6625	4242	83.59
		uniform rate ($\delta = 30\%$)	4178	1698	77.20
100	DM		367248	351102	
	NRM		376892	373726	
	RSSA-Linear	uniform rate ($\delta = 10\%$)	42230	10242	91.31
		uniform rate ($\delta = 20\%$)	37425	2555	83.60
		uniform rate ($\delta = 30\%$)	38597	1044	77.18
	RSSA-Binary	uniform rate ($\delta = 10\%$)	16334	12207	91.26
		uniform rate ($\delta = 20\%$)	7198	2961	83.60
		uniform rate ($\delta = 30\%$)	5846	1311	77.13
	RSSA-Lookup	uniform rate ($\delta = 10\%$)	138395	134081	91.30
		uniform rate ($\delta = 20\%$)	37359	33108	83.60
		uniform rate ($\delta = 30\%$)	17851	13549	77.17

faster than DM) .

The result in Table 4.2 shows two important facts. First, performance depends on the search procedure. Although a complex search procedure runs fast, it spends high computational cost for maintaining the underlying data structure. Thus, in a small network (e.g., $N = 5$) the linear search (RSSA-Linear), which does not require any complex data structure, achieved a better performance than the other search procedures (RSSA-Binary, RSSA-Lookup). Instead, when the network size increases, linear search is no longer the best choice. But, on the other hand, a very expensive data structure e.g., building supported tables in Alias method, does not yield the best performance, either. Consider, as an example, the case $N = 100$. Applying RSSA-Binary and RSSA-Lookup with the same the fluctuation rate $\delta = 20\%$, we can see that their acceptance probability is approximately 83.60%, but RSSA-Binary is nearly 5 times faster than RSSA-Lookup. This is because the update underlying data structure of lookup search method requires too much bookkeeping, indeed the update time of RSSA-Lookup contributes 88% to the total execution time, and is 10 times larger than the RSSA-Binary.

Second, the choice of fluctuation rate δ yielding the best performance is highly dependent on the coupled degree of the reactions. A small value of δ is likely to achieve a better performance when the network size is small (N is small). For example, when $N = 5$, a good choice is $\delta = 10\%$ to 20%. This is because the search time dominates the overall execution time. However, if we increase N, updates largely affect the performance. A larger fluctuation interval would then yield better performance. For example, with $N = 100$, the performance of RSSA-Binary with $\delta = 30\%$ is approximately 1.5 times faster than the case $\delta = 20\%$. Also, RSSA-Lookup becomes 2 times faster when moving from $\delta = 20\%$ to $\delta = 30\%$.

4.3.2 Multiscaled reaction model

The multiscaled reaction model involves a mix of both fast and slow reactions, having the following form.

$$\text{Fast reaction } R_i : A_i \xrightarrow{k_i} A_j$$
$$\text{Slow reaction } R_j : A_i + B_j \xrightarrow{k_j} B_k$$

The chemical species A_i are the only ones to occur in fast reactions, and are therefore named the “fast species”. By contrast, slow reactions involve also other species B_j, named the “slow species”. We assume N fast species A_i and M slow species B_i. The model then is generated in this way. First, we include all the fast reactions $A_i \xrightarrow{k_i} A_j$ for any value of i, j. Then, we add M slow reactions $A_i + B_j \xrightarrow{k_j} B_k$ where A_i and B_k are chosen randomly, while B_j ranges over the M slow species.

In this model, reaction rates of fast reactions are chosen to be much larger than the slow reactions ($k_i \gg k_j$). The initial population of each fast species is set to 1000, while slow species are set to 100. The propensity function is simply the one given by the mass-action kinetics. In our experiment we fix the number of fast species to $N = 5$, and vary the number of slow species M (hence also varying the number of slow reactions) from 50 to 1000. In RSSA, we implemented two fluctuation interval control mechanisms: 1) uniform fluctuation rate and 2) non-uniform fluctuation rate. The uniform rate δ is adjusted between 10% and 20%. In non-uniform fluctuation rate, the fast species are assigned rate $\delta_{fs} = 20\%$ and slow species are assigned $\delta_{ss} = 10\%$. Table 4.3 compares the performance of algorithms applied to multiscaled reaction model.

A slow reaction is formed by combining a slow species and a randomly selected fast species, so we have on average $M/N + 1$ affected reactions which must update their propensities each time a fast reaction fires. Update time is linearly increasing with the number of slow species M. Thus, the update time dominates the total simulation time as M is increasing. This effect is shown in

Table 4.3. From the results we observe that, even with the small network size $M = 50$, roughly 92% of the execution time is spent for update in NRM, and this value is increasing to over 97% with $M \geq 100$. The update time of DM also exposes the same effect in that it contributes roughly 95% to the simulation. The performance of these algorithms is thus rather low. RSSA achieves better performance because it effectively controls the time-consuming updates. In the case $M = 1000$, RSSA-Linear with uniform rate $\delta = 20\%$ is more than 45 times faster than DM, while RSSA-Lookup with non-uniform rate $\delta_{fs} = 20\%$, $\delta_{ss} = 10\%$ is 185 times faster than DM.

From Table 4.3, the acceptance probability of RSSA with non-uniform fluctuation rate $\delta_{fs} = 20\%$, $\delta_{ss} = 10\%$ is between the one for uniform rates $\delta = 20\%$ and $\delta = 10\%$, when the same search procedure is applied. This is because we keep the acceptance probability of fast reactions to the same as the one in uniform rate $\delta = 20\%$, and also increase the acceptance probability of slow reactions. As a result, the performance of RSSA with non-uniform rate $\delta_{fs} = 20\%$, $\delta_{ss} = 10\%$ is better than the one with uniform fluctuation rate $\delta = 20\%$. However, it is not always better than the case $\delta = 10\%$ (for example, see RSSA-Linear and RSSA-Binary), even though the difference is quite small. This is because the search of these algorithms is more expensive than the update. By contrast, RSSA-Lookup uses a fast search procedure, so it needs a better mechanism to control the fluctuation interval. Thus, in case $M = 1000$, RSSA-Lookup with non-uniform fluctuation rate $\delta_{fs} = 20\%$, $\delta_{ss} = 10\%$ is nearly 5% faster than the RSSA-Lookup with uniform rate $\delta = 20\%$, and 10% faster than with uniform rate $\delta = 10\%$.

4.3.3 Gene expression model

The gene expression model is a type of regulatory pathway which plays a key role in the understanding of gene regulation mechanisms and functionality. The result of gene expression is a collection of proteins encoded by the correspond-

Table 4.3: Performance of algorithms on multiscaled reaction model

M	Algorithm		Execution Time (ms)	Update Time (ms)	Acceptance Prob. (%)
50	DM		16280	15062	
	NRM		17740	16987	
	RSSA-Linear	uniform rate ($\delta = 10\%$)	1823	5	90.92
		uniform rate ($\delta = 20\%$)	1976	0	81.43
		non-uniform ($\delta_{fs} = 20\%, \delta_{ss} = 10\%$)	1868	0	83.93
	RSSA-Binary	uniform rate ($\delta = 10\%$)	1662	3	90.98
		uniform rate ($\delta = 20\%$)	1778	0	81.36
		non-uniform ($\delta_{fs} = 20\%, \delta_{ss} = 10\%$)	1700	0	83.96
	RSSA-Lookup	uniform rate ($\delta = 10\%$)	1642	9	90.88
		uniform rate ($\delta = 20\%$)	1799	0	81.33
		non-uniform ($\delta_{fs} = 20\%, \delta_{ss} = 10\%$)	1779	0	83.90
100	DM		29416	27918	
	NRM		32240	31346	
	RSSA-Linear	uniform rate ($\delta = 10\%$)	2241	7	90.90
		uniform rate ($\delta = 20\%$)	2367	0	81.06
		non-uniform ($\delta_{fs} = 20\%, \delta_{ss} = 10\%$)	2304	0	82.79
	RSSA-Binary	uniform rate ($\delta = 10\%$)	1802	11	90.91
		uniform rate ($\delta = 20\%$)	1883	0	81.19
		non-uniform ($\delta_{fs} = 20\%, \delta_{ss} = 10\%$)	1843	0	82.74
	RSSA-Lookup	uniform rate ($\delta = 10\%$)	1809	18	90.94
		uniform rate ($\delta = 20\%$)	1892	0	81.18
		non-uniform ($\delta_{fs} = 20\%, \delta_{ss} = 10\%$)	1885	0	82.89
500	DM		173357	169555	
	NRM		194623	193520	
	RSSA-Linear	uniform rate ($\delta = 10\%$)	4291	64	90.90
		uniform rate ($\delta = 20\%$)	4534	0	79.18
		non-uniform ($\delta_{fs} = 20\%, \delta_{ss} = 10\%$)	4535	0	81.22
	RSSA-Binary	uniform rate ($\delta = 10\%$)	1881	55	90.79
		uniform rate ($\delta = 20\%$)	1971	0	79.07
		non-uniform ($\delta_{fs} = 20\%, \delta_{ss} = 10\%$)	1892	0	81.92
	RSSA-Lookup	uniform rate ($\delta = 10\%$)	1903	67	90.78
		uniform rate ($\delta = 20\%$)	1918	0	78.94
		non-uniform ($\delta_{fs} = 20\%, \delta_{ss} = 10\%$)	1859	0	81.35
1000	DM		377483	371024	
	NRM		404654	403219	
	RSSA-Linear	uniform rate ($\delta = 10\%$)	7557	119	90.67
		uniform rate ($\delta = 20\%$)	8242	4	78.06
		non-uniform ($\delta_{fs} = 20\%, \delta_{ss} = 10\%$)	8079	6	79.89
	RSSA-Binary	uniform rate ($\delta = 10\%$)	2010	110	90.70
		uniform rate ($\delta = 20\%$)	2071	3	78.10
		non-uniform ($\delta_{fs} = 20\%, \delta_{ss} = 10\%$)	2016	4	79.67
	RSSA-Lookup	uniform rate ($\delta = 10\%$)	2049	156	90.61
		uniform rate ($\delta = 20\%$)	1998	2	78.09
		non-uniform ($\delta_{fs} = 20\%, \delta_{ss} = 10\%$)	1914	6	79.77

Table 4.4: Gene expression model

R_1: $G \to G + RNA$	$k_1 = 0.09$
R_2: $RNA \to RNA + P$	$k_2 = 0.05$
R_3: $RNA \to \_$	$k_3 = 0.001$
R_4: $P \to \_$	$k_4 = 0.0009$
R_5: $P + P \to P_2$	$k_5 = 0.00001$
R_6: $P_2 \to P + P$	$k_6 = 0.0005$
R_7: $P_2 + G \to P_2G$	$k_7 = 0.005$
R_8: $P_2G \to P_2 + G$	$k_8 = 0.9$

ing genes. It composes two main consecutive processes: transcription and translation. The transcription initiates when an enzyme called RNA polymerase (RNAP) binds to gene promoter. During the transcription process, the gene is copied to intermediate form called mRNA. In the translation process mRNA will then bind to ribosomes to translate into the corresponding protein.

The 8 reactions shown in 4.4 depict a typical gene expression model. In this table, protein P is encoded by gene G. The intermediate product of transcription is denoted by RNA. The transcription was modelled by reaction R_1 where gene G transcribes to RNA. RNA, after translating to protein P through reaction R_2, will degrade by reaction R_3.

The proteins usually interact to form a dimer P_2 rather than existing in the isolated form. Reactions R_5 and R_6, respectively, model the association and dissociation of dimers P_2. The dimer could bind to gene G to enhance the activation of the gene. Thus this is modelled by reaction R_7, R_8.

In simulating this model, we set the initial population of gene G to $10,000$, while other species are set to 0. We implement the adaptive fluctuation interval control to compare with other mechanisms. The threshold is set to $\lambda = 25$. We dynamically choose between the fluctuation rate δ and a fixed interval size Δ. Table 4.5 compares the performance of the different simulation algorithms.

Table 4.5: Performance of algorithms for Gene Expression Model using mass-action kinetics propensity

Algorithm		Execution Time (ms)	Update Time (ms)	Acceptance Prob. (%)
DM		3128	1922	
NRM		3167	2459	
RSSA-Linear	uniform rate ($\delta = 10\%$)	2444	144	86.77
	uniform rate ($\delta = 20\%$)	2513	30	75.89
	adaptive rate ($\delta = 20\%, \Delta = 5$)	2386	120	86.51
RSSA-Binary	uniform rate ($\delta = 10\%$)	2724	175	86.79
	uniform rate ($\delta = 20\%$)	2520	44	75.85
	adaptive rate ($\delta = 20\%, \Delta = 5$)	2479	123	86.58
RSSA-Lookup	uniform rate ($\delta = 10\%$)	2538	161	86.78
	uniform rate ($\delta = 20\%$)	2583	45	75.86
	adaptive rate ($\delta = 20\%, \Delta = 5$)	2523	140	86.56

From the result Table 4.5, the performance of NRM and DM is nearly the same, although DM is slightly faster than NRM. Although this model is quite small, it also requires a high cost for updating, which contributes 77% to the total simulation time in NRM, while in DM this contributes 62%. Even in this model RSSA-Linear with $\delta = 10\%$ could reduce the update time to only 6%. Hence, its performance is approximately 22% faster than DM and NRM. In this small model, it is easy to see from Table 4.5 that RSSA-Linear is a bit faster than RSSA-Binary and RSSA-Lookup; however, the difference between the performance of these implementations of RSSA is quite small. Second, for this model a narrow fluctuation interval (small value of δ) would yield a high acceptance probability, and thus better simulation time.

Because the population of species involved in this model is quite low (both at beginning and at the stable state), the combination of uniform fluctuation rate and fixed interval size yields the best performance. For example, RSSA-Linear with the combination of uniform rate $\delta = 20\%$ and fixed interval size $\Delta = 5$ is nearly 4% better than RSSA-Linear using only the uniform rate $\delta = 10\%$ which is the best performance achieved while applying uniform rate.

For second experiment, we consider the effects of evaluating the complex propensity function to the update and the total simulation performance. Hence, we modified the propensity function to use the Hill kinetics. This kinetics was first used to model the nonlinear effects of aggregation of the haemoglobin molecules with oxygen in the solution [75]. The Hill equation recently has extensive applications in pharmacology to model the nonlinear relationship in drug-dose response on the target (see e.g. [70] for details). In biology, Hill kinetics has been used to model the mechanism of enzymatic reactions. The Michaelis-Menten law, a well-known model of enzyme kinetics, is a special type of Hill kinetics. Hill kinetics is commonly used to describe the cooperativity of a ligand binding to an enzyme. In this cooperative binding, the binding of a ligand to an enzyme is often enhanced the enzyme operativities if there are already ligands binding to this enzyme. In modelling of gene expression, Hill kinetics has applied to describe the activation controlled in the gene regulation process. For example, in [86], it was used to model the switch-like behavior in the gene expression by protein activation. In our experiment, we use the propensity function with Hill equation which has a general form:

$$g(x) = \frac{x^n}{K^n + x^n} \tag{4.6}$$

where K is threshold constant and n is the steepness parameter (also called Hill coefficient), which is usually non-integer.

The simulation runtime of the gene expression model with Hill kinetics of different simulation algorithms is in Table 4.6. Since evaluating the propensity now requires more computational effort, the performance of DM and NRM is

Table 4.6: Performance of algorithms for Gene Expression model with Hill kinetics propensity

Algorithm		Execution Time (ms)	Update Time (ms)	Acceptance Prob. (%)
DM		9517	8307	
NRM		8188	7480	
RSSA-Linear	uniform rate ($\delta = 10\%$)	3423	615	86.78
	uniform rate ($\delta = 20\%$)	3655	162	75.87
	adaptive rate($\delta = 20\%, \Delta = 5$)	3389	568	86.54
RSSA-Binary	uniform rate ($\delta = 10\%$)	3432	661	86.77
	uniform rate ($\delta = 20\%$)	3613	161	75.91
	adaptive rate ($\delta = 20\%, \Delta = 5$)	3404	562	86.59
RSSA-Lookup	uniform rate ($\delta = 10\%$)	3537	636	86.79
	uniform rate ($\delta = 20\%$)	3720	177	76.00
	adaptive rate ($\delta = 20\%, \Delta = 5$)	3508	617	86.57

roughly 3 times slower than the mass-action kinetics propensity. In the simulation involving the Hill kinetics, the propensity computation contributes 87% of the overall time in DM, and 91% in NRM. By our RSSA simulation method, the update cost of RSSA-Linear with $\delta = 10\%$ is kept nearly at 17% of the total cost, and the overall performance is roughly 2.5 times better than DM. In this experiment, RSSA with a combination of uniform rate and fixed interval size also achieves a better performance than by only using uniform rate. This is because it handles the low population of species better.

4.4 Towards an Optimal Parameter Selection

We have proposed several improvements to RSSA, and shown their efficiency in applying to concrete models in the previous sections. Still, a systematic approach is required to automatically select the optimal parameters e.g., the fluctuation rate vector, in order to optimize performance. The tunable parameters used in RSSA essentially are related to the fluctuation control mechanism and the search procedure. Because these factors are correlated, optimizing both can not be performed by handling them independently. For example, a fluctuation rate used with a specific fluctuation control mechanism could yield an optimal performance for RSSA with a linear search, but that might not the optimal choice when applying other strategies e.g. binary search, lookup search. In fact, the selection of parameters for optimizing RSSA can be regarded as a combinatorial optimization problem. Several global optimization techniques have been developed to tackle this task (see e.g. [58, 149] and references therein). In this section, however, we limit our focus on choosing the parameters for fluctuation control with a given search procedure. Our approach is based on a gradient-like method, called stochastic approximation (SA) [136], to estimate the rate parameter. SA is essentially an iterative algorithm. In each iteration, the parameter is estimated by a similar form of the gradient-based optimization; however, the gradient is approximated by using a simulation instead of using a fixed exact analytic form. The advantage of this method is that it does not require a detailed knowledge of the relationship of the rate parameter and the performance measurement being considered.

Let $T_{RSSA}(r, m, c, a)$ be the measurement of the run time of RSSA, where r is the fluctuation rate vector used by fluctuation control c with a specific search algorithm a, and m is the given simulated model (i.e., a reaction network). $T_{RSSA}(r, m, c, a)$ is regarded as a random value, to be determined by simulation. Our purpose is finding:

$$\min_{r \in \mathcal{R}} \mathbb{E}[T_{RSSA}(r, m, c, a)] \tag{4.7}$$

Hence, the objective is to find the parameter r in the parameter space $\mathcal{R}$ minimizing the expected run time $\mathbb{E}[T_{RSSA}(r, m, c, a)]$.

Starting with initial guess r_0. SA estimates the parameter r_k at iteration k by the form:

$$r_{k+1} = r_k + a_k g_k(r_k) \tag{4.8}$$

where $g_k(r_k)$ is an estimation of the gradient of the performance measurement, and a_k is the (positive) step size. This principle of the estimation is based on local changes of the rate parameter. There are two main implementations of this method: namely, the finite difference stochastic approximation (FDSA) and the simultaneous perturbation stochastic approximation (SPSA) [58, 148]. In the former only one component in parameter vector r_k is perturbed at a time, while in the latter all components of the rate parameter are randomly perturbed.

Let p be the dimension of the rate vector parameter. FDSA estimates the approximated gradient $g_k(r_k)$ is as following. The ith component in the rate parameter r_k is perturbed by a small positive constant c_k hence obtaining $r_k + c_k e_{ki}$ with $1 \leq i \leq p$ where e_{ki} is a unit vector having its i-th component set to 1, while all the other are zero. Then, the ith element in the estimated gradient $g_k(r_k)$ is approximated by

$$g_{ki}(r_k) = \frac{T_{RSSA}(r_k + c_k e_{ki}) - T_{RSSA}(r_k - c_k e_{ki})}{2c_k} \tag{4.9}$$

The number of evaluations of the performance function T_{RSSA} grows linearly with the parameter dimension. Indeed, we need to perform exactly $2p$ such evaluations. Hence, as the number of parameters p becomes large, the cost to reach convergence increases.

SPSA takes advantage over DFSA in estimating the gradient by evaluating the performance T_{RSSA} independently of the dimension of the rate parameter. SPSA approximates the gradient by perturbing all the components by a user specified random p-vector $\Delta_k = (\Delta_{k1}, \ldots, \Delta_{kp})$. The i-th component of the estimated gradient is computing by:

$$g_{ki}(r_k) = \frac{T_{RSSA}(r_k + c_k \Delta_k) - T_{RSSA}(r - c_k \Delta_k)}{2c_k \Delta_{ki}} \tag{4.10}$$

The random vector Δ_k should be chosen so that the $\{\Delta_{ki}\}$ components are independent, symmetrically distributed around 0, and having finite inverse moments. The most common selected distributed satisfied this condition is the Bernoulli ± 1 distribution. The selection of (a_k, c_k) values, and the convergence of the stochastic approximation (in suitable conditions), have been studied in [149].

We applied the SA method discussed above to optimize, for any given search algorithm, the fluctuation rate parameter for the fully connected reaction model. Results are shown in Fig. 4.4. The figure shows that, when increasing the coupled degree of the reaction network, the fluctuation rate should be increased, so to reduce the update time. This agrees with the experiments discussed in the previous section. Note that the optimal rate for each search procedure is different, especially when the coupled degree is high. For example, when $N = 5$ (w.r.t. the coupled degree = 4) the difference in the optimal rates used by search procedures are small i.e. 11.95%, 19.12% and 12.67% for RSSA-Linear, RSSA-Binary and RSSA-Lookup, respectively. The optimal rates are quite similar in the case $N = 10$ (the optimal rate is around 22%). Then, if we continuously increase the coupled degree the optimal rate for each variant of RSSA becomes very different. The optimal fluctuation rate for RSSA-Linear slowly increases with N. Instead, the one for RSSA-Lookup grows much faster. The rate for RSSA-Binary lies somewhere in the middle. For example, with $N = 100$ the optimal rate of RSSA-Linear is 25.27%, while in RSSA-Binary is 37.37%. RSSA-Lookup achieves the best performance with the rate around 57.75%. This can be explained as follows. The lookup search in RSSA-Lookup is very efficient, but the rebuilding of the underlying data structures is rather expensive. Hence, it has to use a large rate to reduce the number of rebuilding steps, at the price of reducing acceptance probability as well.

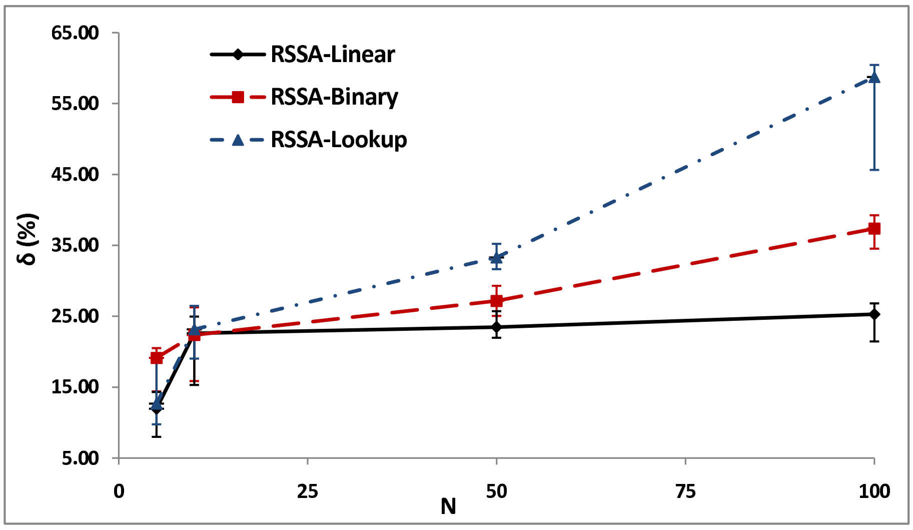


Figure 4.4: Optimal Fluctuation Rate for Fully Connected Reaction Model

4.5 Conclusions

In this chapter we proposed a new generalized algorithm, called RSSA, for doing stochastic simulation. RSSA, in essential, is a rejection-based algorithm. The selection of a reaction firing composes of two steps. RSSA uses a propensity upper-bound to select a candidate reaction. An acceptance-rejection procedure is then used to verify the candidate reaction. We mathematically proved RSSA, performing in this way, produces the same stochastic behaviour as SSA.

Then, we investigated how to improve RSSA, by studying how to tune its performance so to efficiently simulate biochemical reaction systems. First, we experimentally explored different search procedures for implementing the selection of a candidate reaction. The optimal choice ultimately depends on the problem size and complexity of the underlying data structures. Some search procedures e.g., binary search, alias method can obtain a fast search time, but also require data structure which is expensive to update; instead, a simple search method e.g., linear search, does not require any complex data structure, while having a low search performance. According to our experiments, linear search is best used on small models, while more complex methods should be applied on large models. Second, Several search algorithms for implementing the selection of a candidate reactions are proposed. We experiment with their implementations using different network sizes, and discuss the results of such experiments.

Second, we proposed different mechanisms to control acceptance probability of a candidate reaction. The proposed mechanisms run at different levels, in a static or dynamic fashion during the simulation. A dynamic control give more flexibility for controlling the acceptance probability depending on the state of the system, but it also requires more computational effort. We also discussed an approach to automatically select the optimal rate for a given search and fluctuation control mechanism.

In RSSA, the acceptance-rejection procedure is applied to reduce the cost of propensity updates while, in the literature, it has been used for different purposes. We clarify some applications of acceptance-rejection procedure to stochastic simulation with RSSA in the following. The composition rejection SSA (CR-SSA) [147] uses the acceptance-rejection method for improving only the search of the next reaction firing. The search time of CR-SSA is indeed a constant time for long run. Thus, if the network is dense and highly coupled, its update time will contribute a significant portion to the simulation. The *uniformization* technique is proposed in [141] to discrete the time in which it uses a global upper-bound of total propensity instead of upper-bound reaction propensities in RSSA. Hence, the simulation only needs the search and update of reactions without generating the firing time. The discrete time conversion approach is different with RSSA in many aspects. First, while it does not need to generate the reaction firing time, it still requires paying the search and update costs in each simulation step, using exact propensity values. Second, in order to approximate the global upper-bound of total propensity it has to know a *global* upper-bound for the population of all species. This is hard to pre-compute. Indeed, even in the case such upper-bound is known, the upper-bound of total propensity may be several orders of magnitude larger than the actual total propensity e.g. if the system is stiff. The result is the simulation would spend a lot of time to reject the candidate reaction. In the contrast, RSSA controls the upper-bound of each reaction propensity. Furthermore, we can efficiently control the upper-bound of each reaction propensity in runtime through a fluctuation control techniques, as discussed in previous section. This provides a needed flexibility, allowing one to adjust the acceptance probability as desired. The simulation performance thus can be sensibly improves by tuning these parameters. A rejection-based simulation algorithm recently proposed in [40] for simulating the signaling pathways. This simulation algorithm is different with RSSA in some senses. First, it rather applies the rejection step to skip counting all possible combinations of the receptor-ligand binding which is typically a huge number. Second, it exploits a very complex timing scheme to match the exact time. In our RSSA method, the timing scheme is clean and simple, i.e., the Erlang distribution. Furthermore, it is proved that the time also follows the exact distribution.

Further studies are possible to improve the performance of RSSA. For instance, when propensities are given by a user-specified complex function, one needs to devise an efficient way to compute the propensity lower- and upper-bounds. This may be done automatically, or with some help from the user. The impact of this choice still needs to be evaluated. Another research line would focus on using global optimization techniques to fine-tune RSSA performance. This approach would suggest the optimal combination of methods to use, i.e., which search procedure and which control mechanism. Integrating such optimization techniques in simulation is a non trivial task since the time required to run them might negate their benefits. This would indeed require further investigation.

Chapter 5

Rejection-based reaction diffusion

5.1 Introduction

The dynamic behaviour of living cells is indeed dependent on both the reaction and diffusion of molecular species. The significance of diffusion becomes highly important when the diffusion time of species is slower than the reaction time. The biological systems will exhibit inhomogeneities. Furthermore, the cell is highly compartmentalized. Diffusion between sub-compartments formed by localized species significantly magnifies the noise effects on reaction pathways. These thus imply a crucial coupling of reaction and diffusion. Spatial heterogeneity recently has been successful in explaining many experimental observations, e.g., localization of the E. coli cell division [52]. In this situation, the simulation should explicitly take into account the diffusion of the species in reaction networks.

The spatial extensions of SSA have been introduced to simulate reaction-diffusion systems [17]. These methods are based on discretizing the space into subvolumes. The subvolume side length is chosen so that the subvolume is well-mixed. It further assumes that a species in a subvolume only reacts with species in the same subvolume. Hence, the same molecular species in different subvolumes is treated separately. The diffusion of a diffusive species between neighbor subvolumes is modelled as a unimolecular reaction. The kinetics of this enlarged network is mathematically modelled by the reaction-diffusion master equation (RDME). RDME is in fact a spatial extension of CME. It, therefore, is possible to simulate by SSA.

Although RDME, in principle, can be exactly simulated by SSA, a direct application of SSA to sample RDME is often computationally intensive because the number of species and reactions in the model is linearly increased by the number of subvolumes. An efficient implementation for performing stochastic reaction-diffusion simulation is to select the subvolume which contains the next reaction firing, and then retrieve out the next reaction firing in that subvolume. There are many possible combinations for implementing these steps. For example, these two steps can be done in two consecutive DMs in which the first DM searches for which subvolume and the second one is for finding the next reaction in selected subvolume. The Next Subvolume Method (NSM) [45] is an efficient formulation for improving the search of the subvolume. In NSM, the selection of a subvolume is done by exploiting a special priority queue, i.e., the binary heap. The subvolumes are indexed so that the subvolume having smallest putative time is always put on the top of the queue. The search for the subvolume thus is in constant time. Then, the next reaction firing in the selected subvolume is found out by a DM search. Anytime there is a change in the subvolume due to a reaction firing or a diffusive transfer from its neighbor, the priority queue should be maintained to reflect the change. Hence, a significant portion of the simulation time of NSM now is spent for updating the priority queue.

The computational cost for performing stochastic reaction-diffusion simulation is further increasing to ensure a physical consistence and correctness of the spatial discretization. First, for the applicability of SSA in a subvolume, the side length of the subvolume must be chosen much smaller than the so-called Kuramoto length [95]. However, the subvolume side length also should not be chosen arbitrarily small. It has been shown that if the subvolume side length becomes too fine the simulation of RDME yields incorrect and even nonphysical results [12]. There the system is entirely controlled by diffusion, and there is no reaction occurring. In this case, reaction propensities have to be corrected e.g., by some correction factors, to match the results from a particle-based simulation [51]. These correction factors are often rather expensive to compute [48]. Second, the space is often discretized by regular meshes, e.q., the cubical subvolumes, and the rate of the diffusion is often transformed from the Fick's law. The space, however, may be discretized by irregular meshes to deal with the highly complex cell medium, e.g., the cell membrane. In that case the rate of the diffusion reaction also has to be modified by a complex and, of course, computational demanding function [15, 47, 74].

In this chapter we propose a new formulation, called RRD, to alleviate the computational burden of the exact spatial stochastic simulation. RRD improves both the search of a reaction firing in a subvolume and the propensity updates, and hence improving the total simulation performance. More specifically, RRD combines the efficient tree-based search and the methods based on the over-approximation of propensities developed in chapters 3 and 4 to enhance the search and the update. Both the search for a subvolume and then a reaction in the selected subvolume by RRD are using only the over-approximation of propensities. This feature is the highlighted difference with stochastic reaction-diffusion simulation approaches in literature. A candidate subvolume is first found by a tree-based search based on the over approximation of subvolume propensities. Then, given the subvolume, a fast lookup search is conducted to retrieve a candidate reaction in that subvolume. The candidate reaction is committed to fire by a rejection-based mechanism. In case the candidate reaction is rejected, an entirely new candidate subvolume as well as a new candidate reaction in this subvolume have to be selected again. Since the candidate subvolume is discovered by the tree-based search, it scales in a logarithmic way with the number of subvolumes. The update of a tree branch is rarely required unless the population of species in the subvolume jumps out of the fluctuation interval.

Anytime this happens, there are at most two tree branches have to update.

5.2 Reaction-diffusion simulation

5.2.1 Spatial SSA

Assuming the cell volume V is divided into n_V subvolumes, which are denoted by $V_1, \ldots, V_{n_V}$. These subvolumes are further assumed to be spatial homogeneous. Thus, only the population of species in subvolumes are required to keep tracking. Let $X_i^{V_k}(t)$ be population of species S_i in subvolume V_k at time t. The n-vector $X^{V_k}(t) = (X_1^{V_k}(t), \ldots, X_n^{V_k}(t))$ denotes the population vector of subvolume V_k for all $k = 1 \ldots n_V$. Hence, the system state is a n by n_v vector $X(t) = (X^{V_1}, \ldots, X^{V_{n_V}})$ denoting population of each species in each subvolume at time t.

The diffusion of species S_i with diffusion constant D_i from subvolume V_k to its neighbor V_l is explicitly expressed by a unimolecular reaction. That is:

$$S_i^{V_k} \longrightarrow S_i^{V_l} \tag{5.1}$$

the rate of this diffusion reaction for a cubical subvolume with side length h is defined to be D_i/h^2. For an irregular mesh, a correction factor for the diffusion reaction has to be applied [47]. We assume further that there are m_d diffusion reactions in a subvolume.

Let $a_j^{V_k}$ be the propensity of reaction R_j in subvolume V_k. Let $a_0^{V_k}$ be the propensity of subvolume V_k. It is the sum of propensities of all reactions in subvolume V_k, i.e., $a_0^{V_k} = \sum_{j=1}^{m+m_d} a_j^{V_k}$. Let $a_0 = \sum_{k=1}^{n_V} a_0^{V_k} = \sum_{k=1}^{n_V} \sum_{j=1}^{m+m_d} a_j^{V_k}$ be the total propensity of the system.

The spatial stochastic simulation makes a trajectory of RDME by sampling the joint next reaction probability distribution function $p(\tau, j, k|x, t)$ which denotes the probability the reaction R_j inside the subvolume V_k occurring at the next time $t + \tau$ given current state $X(t) = x$ at time t. The reaction firing time is distributed following an exponential distribution $a_0 exp(-a_0\tau)$. The subvolume V_k is selected given the time τ following a discrete probability function $a_0^{V_k}/a_0$. The conditional probability of the reaction R_j firing in subvolume V_k at time τ follows a discrete probability function $a_j^{V_k}/a_0^{V_k}$. The joint next reaction probability $p(\tau, j, k|x, t)$ for the reaction-diffusion process thus has the form:

$$p(\tau, j, k|x, t) = a_j^{V_k} exp(-a_0\tau) \tag{5.2}$$

The sampling of the joint next reaction probability $p(\tau, j, k|x, t)$ is done as follows. First, the firing time τ is generated by sampling the exponential distribution with mean $1/a_0$. Then, two consecutive searches are conducted to find which the subvolume V_k and after that the reaction within selected subvolume V_k with probability $a_0^{V_k}/a_0$ and $a_j^{V_k}/a_0^{V_k}$, respectively.

NSM improves the subvolume search by an efficient formulation. It uses the putative times of subvolumes to select the subvolume. The putative time τ^{V_k} of subvolume V_k is generated following an exponential distribution with mean $1/a_0^{V_k}$ for all $k = 1 \ldots n_V$. These putative times are indexed in a priority queue so that the subvolume having smallest putative time is always on the top of the queue. When searching for the subvolume, the smallest putative time as well as the corresponding subvolume on the priority queue are extracted. The firing time τ is assigned to be this smallest time. Only the search for the next reaction firing is required. In NSM, it is simply found by sampling the discrete probability function $a_j^{V_k}/a_0^{V_k}$.

Given the selected reaction R_j in the subvolume V_k firing at time τ, the system is updated depending on the type of R_j. If it is a biochemical reaction, the population state of species in subvolume V_k is updated i.e., $X^{V_k} = X^{V_k} + v_j$. In case, it is a diffusive species, a subvolume V_l in the neighbors of V_k is randomly selected. In V_k, one species in the population of S_i is removed, while the population of species S_i in V_k is increased by one. After updating the state, the affected reactions in subvolume(s) recompute their propensities to reflect the changes.

5.2.2 Rejection-based reaction-diffusion simulation

The rejection-based reaction-diffusion (RRD) simulation exploits an over approximation of reaction propensities for selecting a reaction in a subvolume. The search of a reaction firing in RRD is composed of two steps: 1) searching for a candidate subvolume by the tree-based search technique (chapter 3) and 2) finding and committing a candidate reaction in the candidate subvolume by the rejection-based technique (chapter 4).

The approximation of reaction propensities is derived by confined the population of each species in a subvolume to a fluctuation interval. Thus, let us assume the population of species S_i in subvolume V_k at time t is confined to a fluctuation interval $[\underline{X_i}^{V_k}, \overline{X_i}^{V_k}]$. The population state X^{V_k} of subvolume V_k is therefore fixed to the interval $[\underline{X}^{V_k}, \overline{X}^{V_k}]$. Because the same species in different subvolumes is, in general, treated differently by reactions it fluctuates in different manners. The fluctuation intervals of the population state of subvolumes thus can be defined by different fluctuation control mechanisms.

Given a fluctuation interval of species in a subvolume we compute the upper-bound and lower-bound of propensity of reactions in that subvolume. Let $\overline{a_j}^{V_k}$ and $\underline{a_j}^{V_k}$, respectively, be the propensity upper-bound and lower-bound of reaction R_j in subvolume V_k. Let $\overline{a_0}^{V_k}$ be the propensity upper-bound of subvolume V_k. $\overline{a_0}^{V_k}$ is, in fact, the sum of propensity upper-bounds of $m + m_d$ reactions in subvolume V_k. Thus, $\overline{a_0}^{V_k} = \sum_{j=1}^{m+m_d} \overline{a_j}^{V_k}$ for all $k = 1 \ldots n_V$. These upper-bounds will be used for selecting a candidate subvolume and, after that, a candidate reaction in that candidate subvolume. Specifically, the subvolume is discovered by a tree-based search on the propensity upper-bound of subvolume. A table lookup search on the propensity upper-bound of reaction is then applied for searching the candidate reaction.

To do that, a tree for holding these subvolume propensity upper-bounds is built. In the tree, the leaves will contain the subvolume propensity upper-bounds $\overline{a_0}^{V_k}$ for $k = 1 \ldots n_V$, while the internal nodes store the sum of its children. Following this way, the tree root will store total sum of subvolume propensity upper-bound value $\overline{a_0} = \sum_{k=1}^{n_V} \overline{a_0}^{V_k}$. In a subvolume, reaction propensity upper-bounds are used to build up tables for a fast lookup search. The lookup search here is chosen to be the Alias method. The probability vector for the Alias method is $\overline{a_j}^{V_k} / \overline{a_0}^{V_k}$.

The search of a reaction firing starts by first searching for a candidate subvolume placing on the leaves of the tree. The running of the search takes a random number in $[0, \overline{a_0}]$, which decides which the left or right tree branch will be discovered. Beginning at the tree root, the search travels down the tree. If the random value is less than the value stored in the left internal node, the search expands the left branch. Otherwise, it chooses the right branch to explore. In case the search chooses the right branch, the random number is adjusted by subtracting its number by the value stored on the right node. The search repeats until a leaf (candidate subvolume) reached.

Having the candidate subvolume, a candidate reaction in that subvolume is taken out by accessing the lookup tables of the Alias method. Essentially, this method requires a random probability value in $[0, 1]$, and returns a reaction R_j corresponding to this input probability.

The candidate reaction in the subvolume V_k is accepted to fire by a rejection-based mechanism. A random value from $[0, \overline{a_0}^{V_k}]$ is generated. If it is less than the actual reaction propensity $a_j^{V_k}$, the reaction is committed. Otherwise, the selection is rejected. In case the reaction is rejected, new selection step is repeated. The lower-bound propensity $\underline{a_j}^{V_k}$ will be used for quickly accepting the candidate reaction, thus without having to always evaluate reaction propensity $a_j^{V_k}$.

The reaction firing time τ is generated by sampling the $Erlang$ distribution in which the shape parameter is the number of trials k until having a reaction accepted, and the rate parameter is the total propensity upper-bound $\overline{a_0}$. We use the convolution technique described in chapter 4 for doing this task.

Knowing the reaction R_j firing at time τ in the subvolume V_k, the population states of the affected subvolumes are updated. If a biochemical reaction fires, only the population of species involved in the current subvolume V_k is updated i.e., $X^{V_k} = X^{V_k} + v_j$. In contrast, if a diffusive transfer is selected, a random destination subvolume V_l in the neighbours of V_k is taken. The population of corresponding diffusive species S_i in both of these subvolumes is updated. In the uncommon case in which the population state of a subvolume caused by the reaction firing jumps out of the assigned fluctuation interval, a new fluctuation interval should be redefined. The reaction propensity upper-bounds are recomputed and the tree is updated as well. However, only at most two tree branches updates are required because at any time maximum two subvolumes have to update by a reaction firing.

5.2.3 The RRD algorithm

The detailed steps of RRD is listed in Alg. 6. We first define a fluctuation interval $[\underline{X}^{V_k}, \overline{X}^{V_k}]$ for the population state X^{V_k} of each subvolume V_k. Note that we could use different fluctuation intervals for subvolumes. We then compute the lower-bound propensity $\underline{a_j}^{V_k}$ and upper-bound propensity $\overline{a_j}^{V_k}$ for each reaction R_j for $j = 1 \ldots m + m_d$. We then compute the upper-bound subvolume propensity $\overline{a_0}^{V_k}$ for $k = 1 \ldots n_v$. These upper-bound subvolume propensity values will be stored in a tree structure supporting for the tree-based search, while upper-bound reaction propensity values are used to build tables for the Alias lookup.

The main simulation loops until the time t passes over a predefined simulation time T_{max}. A simulation step consists of three steps: 1) selecting a reaction firing in a subvolume, 2) generating firing time and 3) updating the system.

The selection of a reaction firing is repeated until flag $accepted$ is set to **true**.

Algorithm 6 RRD procedure

```
 1: for all subvolume $V_k$ where $k = 1 \to n_v$ do
 2:     define fluctuation interval $[\underline{X}^{V_k}, \overline{X}^{V_k}]$ for population state $X^{V_k}$
 3:     compute propensity lower-bound $\underline{a_j}^{V_k}$ and upper-bound $\overline{a_j}^{V_k}$ for $j = 1 \ldots m + m_d$
 4:     compute subvolume propensity upper-bound $\overline{a_0}^{V_k}$
 5:     build supporting tables for Alias method for reactions in subvolume $V_k$
 6: end for
 7: build a tree for upper-bound subvolume propensity $\overline{a_0}^{V_k}$ for all $k = 1 \ldots n_v$
 8: while $t < T_{max}$ do
 9:     set $u = 1$
10:     set $accepted = $ false
11:     repeat
12:         generate four random numbers $r_1, r_2, r_3$ and $r_4$ from uniform distribution $U(0, 1)$
13:         apply tree-based search for finding candidate subvolume $V_k$ with search value $r_1\overline{a_0}$
14:         apply Allias method for lookup a candidate reaction $R_j$ in subvolume $V_k$ with proba-
            bility $r_2$
15:         if $r_3 \leq (\underline{a_j}^{V_k}/\overline{a_j}^{V_k})$ then
16:             $accepted = $ true
17:         else
18:             evaluate $a_j$ with current state $X^{V_k}$
19:             if $r_3 \leq (a_j^{V_k}/\overline{a_j}^{V_k})$ then
20:                 $accepted = $ true
21:             end if
22:         end if
23:         set $u = u \cdot r_4$
24:     until $accepted$
25:     set transition time $\tau = (-1/\overline{a_0}) \ln(u)$
26:     update time $t = t + \tau$
27:     if $R_j$ is a biochemical reaction then
28:         update population state of subvolume $V_k$ by $X^{V_k} = X^{V_k} + v_j$
29:     else
30:         if $R_j$ is a reaction diffusion of species $S_i$ then
31:             get a neighbor subvolume $V_l$
32:             remove one from population of species $S_i$ in subvolume $V_k$
33:             add one to population of species $S_i$ in subvolume $V_l$
34:         end if
35:     end if
```

```
36:     **for all** affected subvolume $V_k$ **do**
37:         **if** $X^{V_k} \notin [\underline{X}^{V_k}, \overline{X}^{V_k}]$ **then**
38:             define a new fluctuation interval $[\underline{X}^{V_k}, \overline{X}^{V_k}]$
39:             compute propensity lower-bound $\underline{a_j}^{V_k}$ and upper-bound propensity $\overline{a_j}^{V_k}$
40:             build supporting tables for Alias method in subvolume $V_k$
41:             propagate the change in subvolume $V_k$ in the path from it to the tree root
42:         **end if**
43:     **end for**
44: **end while**
```

A trial composes of three consecutive steps. First, a candidate subvolume V_k is found by applying the binary search on the upper-bound subvolume propensity tree built in preparation with the search value is $r_1\overline{a_0}$ where r_1 is a random value from $U(0,1)$. Then, a candidate reaction R_j in that subvolume is retrieved by applying the Alias lookup method. The lookup requires a random value $r_2 \sim U(0,1)$. Third, the candidate reaction is subjected for an acceptance-rejection procedure. The trial is successful if the actual reaction propensity $a_j^{V_k}$ is greater than $r_3\overline{a_j}^{V_k}$ where r_3 is a random value from $U(0,1)$. We quickly accept the candidate reaction without evaluating the actual reaction propensity if $\underline{a_j}^{V_k} \geq r_3\overline{a_j}^{V_k}$. In the other case, the selection is rejected.

The reaction firing is generated by sampling the $Erlang$ distribution with rate parameter $\overline{a_0}$ and shape parameter is the number of trials until that reaction is accepted. We use the convolution method in sampling the $Erlang$ distribution. For each trial, the variable u is continuously updated to be $u = u * r_4$ where r_4 is a random value generating from $U(0,1)$. Then, the firing time τ of the reaction firing is computed as $\tau = (-1/\overline{a_0})\ln(u)$.

Knowing the reaction R_j in subvolume V_k and its firing time τ, the system is updated depending on the type of the reaction. If the reaction R_j is a biochemical reaction, only population state of this subvolume is updated. In case R_j is a diffusion of species S_i, a random neighbor subvolume V_l is taken. The population of species S_i in V_k is removed by one, while its population in V_l is added one.

The next simulation loop is executed without updating the affected reaction propensities in the subvolume if its population state is still confined in the fluctuation interval. In other case, a new fluctuation interval has to be redefined. The new lower-bound propensity and upper-bound propensity of reactions have to be computed. Then, the upper-bound subvolume propensity is recomputed as well as the tree branch from root to that subvolume is updated to reflect the change. The tables for the Alias lookup used inside the subvolume also have to construct according to the new upper-bound propensities.

5.2.4 Correctness of the RRD algorithm

We prove that RRD selects a reaction firing with the joint probability function $p(\tau, j, k)$. The statement is stated in the Proposition 4.

Proposition 4. *RRD is exactly sampling RDME by selecting a reaction R_j in subvolume V_k to fire at time τ following the joint probability density function $p(\tau, j, k) = a_j^{V_k} exp(-a_0\tau)$.*

Proof. Let $Pr(R_j, V_k)$ be the probability a candidate reaction R_j in subvolume candidate V_k is selected and accepted to fire. We factorize $Pr(R_j, V_k)$ by the chain rule.

$$\begin{aligned}\mathrm{Pr}(R_j, V_k) &= Pr(V_k) \cdot Pr(R_j|V_k) \\ &= \frac{\overline{a_0}^{V_k}}{\overline{a_0}} \cdot \frac{\overline{a_j}^{V_k}}{\overline{a_0}^{V_k}} \cdot \frac{a_j^{V_k}}{\overline{a_j}^{V_k}} \\ &= \frac{a_j^{V_k}}{\overline{a_0}} \end{aligned} \tag{5.3}$$

The derivation in the Equ. 5.3 is using three facts. First, a candidate subvolume V^k is selected with probability $\overline{a_0}^{V_k}/\overline{a_0}$. Second, the reaction R_j in that subvolume is selected with probability $\overline{a_j}^{V_k}/\overline{a_0}^{V_k}$. And, last the candidate reaction is accepted to fire with probability $a_j^{V_k}/\overline{a_0}^{V_k}$.

Now, let $Pr(R,V)$ be the probability an arbitrary reaction R in an arbitrary subvolume V is selected and accepted to fire. We have:

$$\begin{aligned} \Pr(R,V) &= \frac{\sum_{k=1}^{n_V}\sum_{j=1}^{m+m_d} a_j^{V_k}}{\overline{a_0}} \\ &= \frac{a_0}{\overline{a_0}} \end{aligned} \tag{5.4}$$

Thus, the probability the reaction R_j in subvolume V_k is selected to fire given an arbitrary reaction R in an arbitrary subvolume V is selected as:

$$\begin{aligned} \Pr(R_j,V_k|R,V) &= \frac{\frac{a_j^{V_k}}{\overline{a_0}}}{\frac{a_0}{\overline{a_0}}} \\ &= \frac{a_j^{V_k}}{a_0} \end{aligned} \tag{5.5}$$

Let τ be the firing time of the accepted reaction R_j in subvolume V_k. It is indeed exponential distributed with rate a_0, i.e., $Pr(\tau) = a_0 \cdot e^{-a_0\tau}$. It is derived from a similar proof provided in Proposition 3, so we do not repeat it here.

Hence, RRD selects a reaction R_j in subvolume V_k to fire at time τ following a joint probability density function:

$$\begin{aligned} \Pr(\tau,j,k) &= (\frac{a_j^{V_k}}{a_0})(a_0 \cdot e^{-a_0\tau}) \\ &= a_j^{V_k} \cdot e^{-a_0\tau} \end{aligned} \tag{5.6}$$

□

5.3 Experimental results

We implement and compare three algorithms including: TreeRD, NSM and RRD. The first implementation is TreeRD which is a variant of spatial SSA in

Table 5.1: Summary of models for reaction-diffusion simulation

Model	Species	Diffusive Species	Biochemical Reactions
cAMP activation of PKA model	6	1	6
Multiscaled reaction model	N + M	N	N(N-1) + M

which the subvolume is discovered by a tree-based search. The tree is built using the subvolume propensities. The reaction firing inside a subvolume is selected by a direct linear search. The second algorithm is an implementation of NSM. We use a binary heap to maintain the priority queue of subvolume putative times. The last algorithm is an implementation of our formulation RRD. All these simulation algorithms are implemented in Java and run on Intel i5-540M processor. The simulation was done after 10^7 steps. The simulation data are recorded for 10^6 steps. The experimental result is averaging over 100 runs. All initializations, which is not a part of simulation loop, are excluded from the calculation.

We report the performance of algorithms on two biochemical reaction models. 1) The cyclic adenosine monophosphate (cAMP) activation of protein kinase A (PKA), and 2) Multiscaled reaction model. The table 5.1 summarizes the properties of simulated models. The first model is a real world model which is used to demonstrate the improvement of RRD. In this model, we experimentally validate the results of the tested algorithms. Then, we show the performance improvement of our formulation. The second model is an artificial model we use to benchmark the simulation performance in different settings. We compare the performance of algorithms by increasing both the number of reactions and subvolumes. The performance of these algorithms now is dependent on two factors: the search of a reaction firing in a subvolume and update of the affected reactions in subvolumes. According to our experiments, our new formulation the simulation performance dramatically outperforms over the tested algorithms.

Table 5.2: cAMP activation of PKA model

R_1: $PKA + 2cAMP \rightarrow PKAcAMP_2$	$k_1 = 8.696 \cdot 10^{-5}$
R_2: $PKAcAMP_2 \rightarrow PKA + 2cAMP$	$k_1 = 0.02$
R_3: $PKAcAMP_2 + 2cAMP \rightarrow PKAcAMP_4$	$k_1 = 1.154 \cdot 10^{-4}$
R_4: $PKAcAMP_4 \rightarrow PKAcAMP_2 + 2cAMP$	$k_1 = 0.02$
R_5: $PKAcAMP_4 \rightarrow PKAr + 2PKAc$	$k_1 = 0.016$
R_6: $PKAr + 2PKAc \rightarrow PKAcAMP_4$	$k_1 = 0.0017$

5.3.1 cAMP activation of PKA model

The cAMP activation of PKA is a part of highly prevalent mammalian signaling pathways that translates an extracellular message into an intracellular response [41, 85]. The cAMP is a second messenger forming when the membrane enzyme adenylyl cyclase is activated. It then goes on activating specific proteins in which an important class is the protein kinase A (PKA). PKA is a tetrameric holoenzyme, consisting of two regulatory subunits (PKAr) and two catalytic subunits (PKAc). PKA is normally inactive in which the regulatory units blocks the catalytic units. The binding of two molecules cAMP to specific locations on the regulatory units of PKA causes the dissociation between the regulatory and catalytic subunits. It thus activates the catalytic units and enables them to phosphorylate substrate proteins. These steps are detailed in Table 5.2.

For running simulation, the space is divided into $n_v = 100$ cubical subvolumes. The diffusion constant of species cAMP is $D_{cAMP} = 300$. The diffusion of all other species is set to zero. At the beginning, there are $30,000$ cAMP molecules placed at the top-left corner of the space, and $30,000$ PKA molecules are uniformly distributed across the space. The population of all other molecules is set to zero.

In Figure 5.1, we plot the average population of three molecules cAMP, PKA

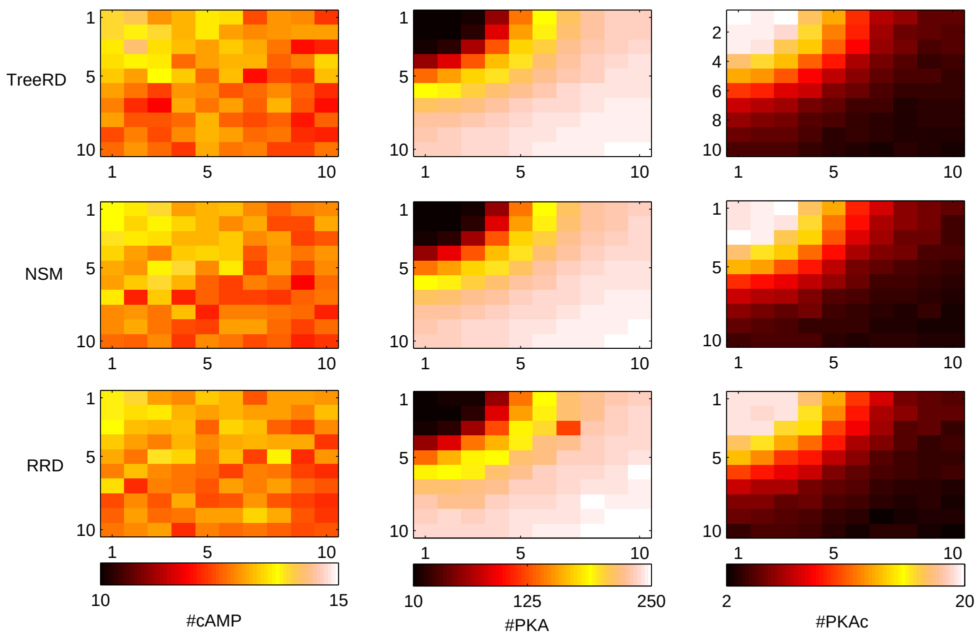


Figure 5.1: Average population of species in cAMP activation of PKA model by algorithms

and PKAc, respectively, over the space at the end of the simulation. The figure shows a strong agreement in the average population of species by simulation algorithms, and thus experimentally confirms the correctness of RRD.

Table 5.3 presents in detail the computational costs for simulation algorithms. In this table, we record the search time which is the time for finding a subvolume and a reaction firing in that subvolume, the update time which is the time required for updating the affected reactions and reflecting the changes to the underlying data structures, and the total simulation time which composes of search time, update time and all other tasks (e.g., random number generation).

From Table 5.3, we see an important fact that the update time contributes a significant portion to the total simulation time. For example, the update cost of NSM contributes up to 80% of its simulation time, while the search time is only 4%. Although the search time for the next reaction firing in the subvolume by NSM is the best, the expensive update negates its advantage. The result is

Table 5.3: Simulation time for cAMP activation of PKA model

Algorithm	Search Time (ms)	Update Time (ms)	Total Time (ms)
TreeRD	1717	17652	24971
NSM	1091	21028	26345
RRD	4135	8825	18237

the performance of NSM is the worst. TreeRD reduces the update time a bit to roughly 70% of its total simulation time. The simulation time of TreeRD is thus slightly better (about 5% faster) than NSM. In this model, RRD yields the best performance even though the search time of RRD is worst (about 3.8 times slower than the search time of NSM). By expoiting the over-approximation of propensities, RRD does not require to update the system at any time after a reaction firing. The update is rarely taken only as needed, hence substantially reducing the update time. The update of RRD is about 2.8 times faster than the update time of NSM. In this experiment, the update time of RRD is reduced to 48% of its total simulation time. As a result, the total simulation time of RRD is roughly 30% and 27% faster than NSM and TreeRD, respectively.

5.3.2 Multiscaled reaction-diffusion model

The multiscaled reaction-diffusion model consists of N fast species A_i and M slow species B_i. The reactions are also separated into fast reactions and slow reactions. A fast reaction is involving fast species A_i only, while a slow reaction involves both slow species B_j and fast species A_i. To form a slow reaction, a fast species is randomly selected in the collection of N fast species. The product of a slow reaction is a random species from the slow species collection. The reaction rate of fast reaction is chosen many times faster than the slow reactions ($k_i \gg k_j$). In this model, The space is further divided into n_v subvolumes. The fast species are be able to move in space, while the slow species are unmovable. Thus, in a subvolume we have N more diffusion reactions.

$$\begin{aligned}
&\text{Fast reaction } R_i : A_i \xrightarrow{k_i} A_j \\
&\text{Slow reaction } R_j : A_i + B_j \xrightarrow{k_j} B_k \\
&\text{Diffusion reaction } R_d : A_i^{V_k} \xrightarrow{k_d} A_i^{V_l}
\end{aligned}$$

In this experiment, we focus on the effects of search and update cost to simulation performance of algorithms. The search is examined by increasing the number of subvolumes N_v in which N_v is adjusted from 100 to $4,000$ subvolumes. We investigate the effect of update by changing the number of slow species M from 10 to 500. In this model, the number of fast species is fixed $N = 5$. At beginning of the simulation, in each subvolume the initial population of fast species is set to $1,000$ and slow species is 100. Since the aim of this experiment concentrates on the performance of algorithms we do not present the data obtained by the simulation algorithms here, although they have shown a strong agreement. Figure 5.2 shows the detailed simulation performance of algorithms on this model.

Figure 5.2a) compares the search time of three algorithms. Although the search time of TreeRD and NSM with small models is slightly better than RRD, it is not scaled well when increasing the model size. In fact, the search time of TreeRD and NSM sharply increase while adjusting M and N_v from small to large. For example, the search time of NSM for $M = 500, N_v = 4000$ is about 24 times slower than the case $M = 10, N_v = 100$. The search time of RRD also increases by increasing the model size but with smaller rate. The search time of RRD for $M = 500, N_v = 4000$ is only 1.9 times the case $M = 10, N_v = 100$. The result is the search of RRD is 3 times faster than the search of NSM with the same model configuration $M = 500, N_v = 4000$.

In Figure 5.2b) the update time exhibits the same behavior as the search when

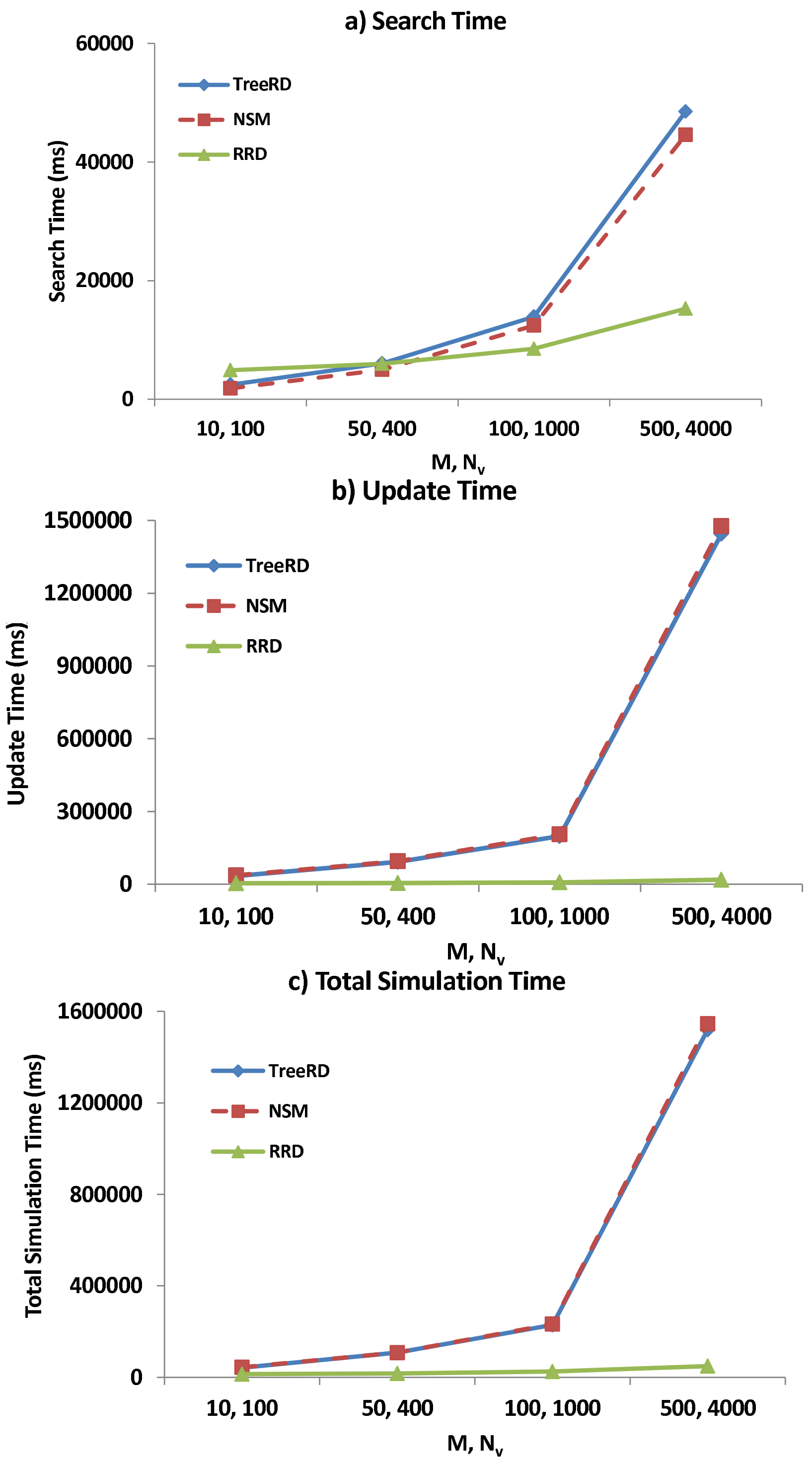


Figure 5.2: Simulation time for multiscaled reaction-diffusion model

increasing the model size. With large model, the update cost becomes extremely expensive. For example, the update time of NSM for $M = 500, N_v = 4000$ is 40 times slower than the case $M = 10, N_v = 100$. RRD handles the update better than TreeRD and NSM. The update time of RRD for $M = 500, N_v = 4000$ is nearly 80 times faster than the NSM and TreeRD.

The total simulation time of three algorithms is shown in Figure 5.2c). From the figure, the performance of TreeRD and NSM is nearly the same, and the performance of RRD is the best for all the cases. Even for small model with $M = 10, N_v = 100$ the performance of RRD is roughly 3 times faster than NSM, TreeRD. By exploiting the efficient search and update, the computational time of RRD is extremely reduced when simulating for large models. In this experiment, the simulation runtime of RRD is around 30 times faster than TreeRD, NSM.

5.4 Conclusion

In this chapter we proposed a new formulation, called RRD, for stochastic reaction-diffusion simulation. RRD combines the over-approximation of reaction propensity and the efficient tree-based search for selecting a reaction firing. The selection of a reaction firing in a subvolume composes of three steps. First, a subvolume is discovered by an efficient tree-based search. Then, a candidate reaction in the selected subvolume is selected by a table lookup. The key point in selecting the subvolume and reaction of RRD is both of these steps are using the over-approxiamtion of reaction propensity. Finally, the candidate reaction is verified to fire based on a rejection-based mechanism. The actual reaction propensity is only required to evaluate at this verification step. These features of RRD is useful for simulating large model for both search and update the system.

Chapter 6

Rare event probability estimation

6.1 Introduction

In this chapter we delve into the problem of performing a statistical analysis of targeted event of interest, such as having a high the population of a specific protein after a determined simulation time. Depending on the event to be studied, a large number of simulation runs may be required to achieve reasonable statistical accuracy. Indeed, the task becomes increasingly harder when considering *rare events*, which occur only with a very small probability. Despite these events being rare, the investigation of such events may be rather important in the study of e.g. the reliability and robustness of a given biochemical system. The occurrence of a rare event could lead the system into an abnormal state, possibly leading to large macroscopic consequences such as the development of a disease e.g., cancer. For example, the epigenetic changes, in amongst other factors, which play important roles in the development of cancer, inactivate tumor suppressor genes and then cause normal cells to be transformed into cancer cells. If the immune system fails to recognize such changes and induce apoptosis, the tumor can spread to healthy cells. As a result, the cancer can grow, possibly causing severe problems to the living organism.

The conventional stochastic simulation, e.g., SSA, for such a task would be to simulate many trajectories and counting the number of the successful ones.

Rare events make this approach infeasible since a prohibitively large number of trajectories would need to be generated before the estimation becomes reasonably accurate. Hence, it is important to devise a method for efficiently producing many evolution samples showing the event of interest. Sufficient information about the rare event could shed light in understanding the developing patterns which lead to the formation of the event.

In this chapter we contribute to the study of rare event simulation by proposing a new simulation algorithm, called *sSSA*, for increasing the frequency of a rare event without otherwise affecting the system behavior. Essentially, ours is an algorithm which *encourages* the evolution of the system so that the target event becomes more likely, yet in such a way that allows one to recover an estimate for the target event probability in the unbiased system. More in detail, our algorithm follows a multi-stage strategy where the system state is divided into nested subsets corresponding to levels that a given trajectory must pass through to reach the desired event. The algorithm works by progressively generating a set of trajectories, and filtering out those which do not reach a given level. The successful trajectories are then used as the basis for a new simulation, generating a new set of trajectories. Then the process is repeated, filtering the new generation according to an higher level, and so on. This is the fundamental idea behind the *multilevel splitting*(see [57, 67, 71, 99, 101] for detailed reviews and discussions). An advantage of this approach is that, while the filtering biases the simulation outcome, the algorithm does not change the reaction rates of their propensities in any way. In this way, we can account for the bias, and still claim the results to be relevant to the model which is being studied. The same idea of multilevel splitting was successfully applied to biochemical networks to calculate the reaction rate constant of the transition between given stable states [4]. However, it requires to fixed the levels before simulating instead of automatic levels chosen in our approach. In the context of rare event simulation there is a different approach based on *importance sampling* [139], where the underlying probability measure of the reaction events, for example, the reaction propensities [98, 98], is manipulated and recovered by multiplying with the so-called likelihood ratio, was introduced to increase the frequency of a rare event. The system, however, is sampling with a different probability distribution, thus it should not be regarded as valid representation of the actual system behavior.

6.2 Problem setting

Let $\Omega \subseteq \mathbb{N}^n$ be the system state space, ranged over by X. Let E_0 and E, respectively, be different subsets of Ω. We want to study the probability of reaching the state $X(t) \in E$ given an initial state $X(0) = x_0 \in E_0$, for some time t bounded by a constant stopping time: $t \leq T_{max}$. In other words, given an event E, we want to compute its reachability probability $\mathbb{P}(E)$. To help intuition, consider the case where the event E corresponds to species S_i having a large population (greater than some threshold λ). The probability to compute $\mathbb{P}(E)$ can then be explicitly expressed as $\mathbb{P}(\exists t \leq T_{max}.\, X_i(t) \geq \lambda |\, X(0) = x_0)$.

Denote with T_E the first time the system hits the event E, i.e., $T_E = inf\{t \geq 0 : X(t) \in E\}$. Our goal is then to efficiently estimate the probability:

$$\gamma = \Pr(T_E \leq T_{max}) = \mathbb{E}[\mathbb{1}_{T_E \leq T_{max}}(X)] \tag{6.1}$$

where $\mathbb{1}_{T_E \leq T_{max}}(X)$ is the indicator function. It returns 1 if $X(T_E) \in E$ s.t. $0 \leq T_E \leq T_{max}$, or 0 otherwise.

We could, in principle, compute γ exactly as in Eq. 6.1 by studying the time evolution of the system. For a well-mixed biochemical system, an exact definition of its evolution is provided by the chemical master equation (CME). However, although the CME completely determines the time evolution of the system, it is hard to solve analytically whenever the state space is not very small. In most cases, stochastic simulation is usually adopted to estimate the probability above. SSA can be used for estimating γ, following Eq. 6.1. This is done by just sampling N i.i.d. trajectories $\{X^i\}_{i=1}^N$ by running SSA from the initial state up to stopping time T_{max}. Each trajectory X^i visits a finite number of states, so checking whether X^i hits the event E is straightforward. We can then estimate the preferred probability $\hat{\gamma}$ as:

$$\hat{\gamma} = \frac{\sum_{i=1}^{N} 1\!\!\mathrm{I}_{T_E \le T_{max}}(X^i)}{N} \tag{6.2}$$

In order to understand how accurate the estimator $\hat{\gamma}$ is, that is how close it is to the actual value γ, we need to study its accuracy as well. By the central limit theorem, $\hat{\gamma}$ approaches a normal distribution $\mathcal{N}(\gamma, \sigma^2/N)$ as N is large enough, where σ^2 is the variance of process X. In other words, we have the relation: $\mathbb{E}[\hat{\gamma}] = \gamma, Var(\hat{\gamma}) = \sigma^2/N$. Although σ^2 is unknown in general, it can be estimated by the (unbiased) sample variance s^2

$$s^2 = \frac{\sum_{i=1}^{N} (1\!\!\mathrm{I}_{T_E \le T_{max}}(X^i) - \hat{\gamma})^2}{N-1} \tag{6.3}$$

To quantify the accuracy of an estimator the *relative error* can be used. It is given by

$$RE = \frac{\sqrt{Var(\hat{\gamma})}}{\mathbb{E}[\hat{\gamma}]} = \frac{\sigma}{\gamma\sqrt{N}} \tag{6.4}$$

The estimation of the above is therefore approximated by $RE \approx s/\hat{\gamma}\sqrt{N}$.

Further, a rough analysis of Eq. 6.4 can be obtained whenever the probability γ is small by noting that we have the approximation $\gamma \cdot (1-\gamma) \approx \gamma$. Thus it yields $Var(\hat{\gamma}) = \gamma \cdot (1-\gamma)/N \approx \gamma/N$, in which $\gamma \cdot (1-\gamma)$ appears as the variance of the Bernoulli variable $1\!\!\mathrm{I}_{T_E \le T_{max}}(X^i)$. The relative error so could be approximated as $RE \approx \sqrt{\gamma/N}/\gamma = 1/\sqrt{N \cdot \gamma}$. From this we can see that in order to reach a given relative error RE, we need to perform at least $1/\gamma RE^2$ trajectory simulations. For instance, if $\gamma = 10^{-6}$ and we want $RE = 1\%$, we need to run roughly $N = 10^{10}$ simulation runs, which seems expensive to perform. Hence, using a trivial SSA sampling to estimate the probability of rare events appears inefficient.

6.3 Splitting for rare event simulation of reaction networks

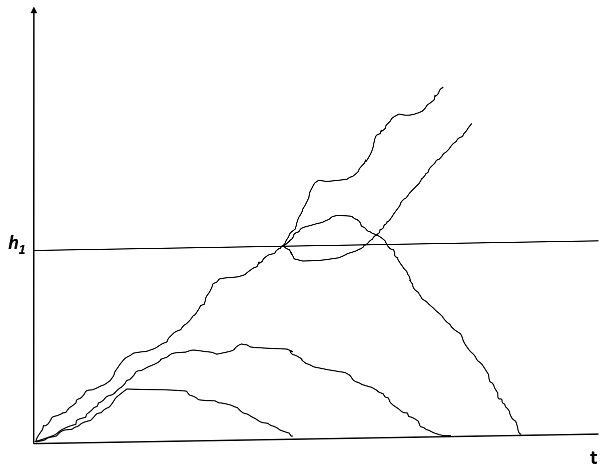
By the discussion in the previous section, estimating the probability of reaching a rare event E by generating many trajectories using SSA is inefficient. The vast majority of such traces will miss E, hence we would need to generate a very large number of trajectories before we can achieve a reasonable accuracy. sSSA improves the efficiency of the simulation by retaining only trajectories which is more likely to reach E, while filtering out the unlikely ones. The promising trajectories will be split into a number of trajectories. It thus needs only a modest number of starting trajectories to estimate the wanted probability with good accuracy. Of course, such estimate is biased. However, we can remove the bias by correcting the estimate using a suitable factor.

6.3.1 Splitting approach

The fundamental of multilevel splitting approach is dividing the state space Ω into some nested subsets $\Omega \supset E_1 \supset \cdots \supset E_L \supset E_{L+1} = E$. This is done so that the probability that a trajectory reaches E_l given that it reaches E_{l-1} is significantly higher than the probability of directly reaching rare event E from the initial state $x_0 \in E_0$. For our purposes we shall assume that these sets can be expressed as $E_l = \{X : \exists t_l \leq T_{max} \wedge h(X(t_l)) \geq h_l\}$, for some *levels* $h_1 < \cdots < h_L$ and a *level function* $h : \Omega \mapsto \mathbb{R}$. Simulation is then applied to estimate all these conditional probabilities i.e., to estimate $p_l = \mathbb{P}(E_l|E_{l-1})$. Finally, an estimator for the probability of the rare event $\mathbb{P}(E)$ is achieved using the chain rule, by letting $\tilde{\gamma} = \prod_{l=1}^{L+1} p_l$.

To estimate the probability of reaching event E given the fixed stopping time T_{max} the *fixed splitting* variant [57] requires to predefine a level sequence $h_1, ..., h_L$. Then, it proceeds as follows. We start with a number of trajectories, say N, from time $t = 0$ at a given initial state $X(0) = x_0 \in E_0$. The trajectories reaching E_1 are kept, while those failing to reach that level are discarded. For

Figure 6.1: Fixed splitting procedure with level h_1 and splitting factor $s = 3$

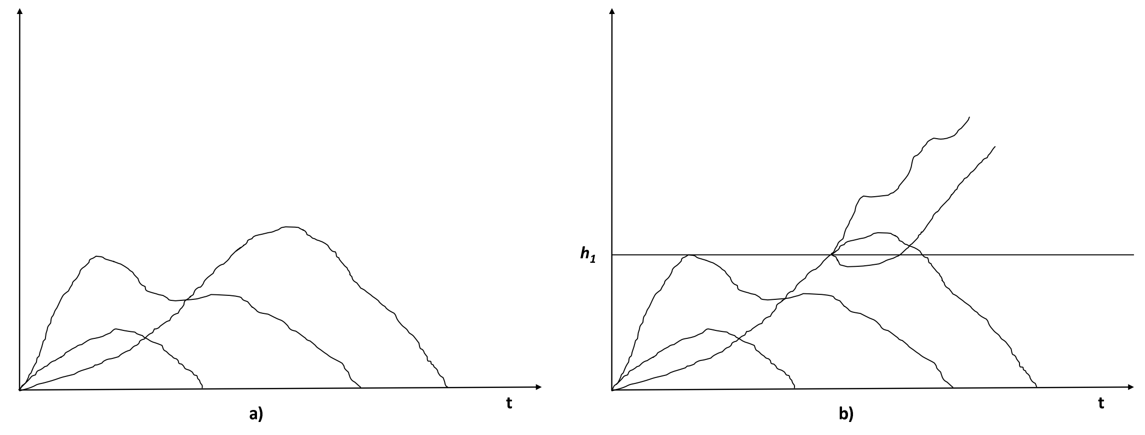


each trajectory we keep, we consider its first entrance state, that is $(X^i(t_1), t_1)$, which falls into E_1. Starting from that entrance state (and time), we again apply simulation to generate s new trajectories. All the trajectories so generated are used to estimate $\mathbb{P}(E_2|E_1)$, and the process is repeated. The constant s is termed the *splitting factor*. The splitting of a trajectory is depicted in Fig. 6.1, which illustrates just one level h_1, and a splitting factor of $s = 3$.

Although fixed splitting variant is unbiased, its accuracy and efficiency are very sensitive to the choice of the parameters. The estimator, indeed, strongly depends on the level sequence and splitting factor. If reaching the next level is unlikely, all trajectories then will probably be discarded. On the other hand, in case reaching the next level is highly probable and the splitting factor is large, the number of trajectories will explode exponentially with the levels.

The *adaptive* multilevel splitting [30] solves the choosing the parameters which does not require an *a priori* choice of levels E_l and splitting factor s. Instead, levels will be defined during its execution. More concretely, to avoid the possibility of trajectories being extinguished, it shall choose intermediate

Figure 6.2: Adaptive choice of level in multilevel splitting, with an ensemble of $N = 3$ trajectories and $k = 1$ trajectory to be kept. In a) 3 trajectories are generated and in b) the 2nd quantile is used as the next level.



levels so that there are always k trajectories reaching the next level. In initial, N trajectories are simulated in which k trajectories with highest value reached by the level function $h(-)$ are kept. In the next stage $N - k$ new trajectories are obtained by prolonging the trajectories reaching the level. These trajectories are merged with the retained trajectories. Then the selecting of k trajectories is repeated. In this way, it keeps the reaching probability $\mathbb{P}(E_l|E_{l-1})$ close to a fixed probability k/N. Hence, it guarantees that they will not be extinguished, and further avoid their exponential explosion as well. We illustrate the above idea in Fig. 6.2. There, we generate an ensemble of $N = 3$ trajectories. For each of them, we compute the maximum level of h which is reached. Then, the level h_1 is picked so to be at the start of the $(k + 1)$th quantile i.e. such that exactly k traces go beyond that level ($k = 1$ in the figure).

Although adaptive multilevel splitting chooses the next level without any a priori knowledge, it introduces bias to the estimated probability of the event of interest. Due to the bias, the adaptive approach could underestimate the probability of the event when it is very small [30].

sSSA takes advantage over adaptive multilevel splitting by ensuring absence of *bias* during the estimation at each stage. sSSA estimates p_l by an ensemble of N trajectories, which are further divided into two disjoint, independent sets of simulation runs N_1 and N_2. More in detail, the N_1 group is used to choose the next level h_l, while the N_2 group is used to estimate the conditional probability $P(E_l|E_{l-1})$. Using the same trajectories for both purposes makes it hard to justify that estimator for the conditional probability is unbiased, since the event E_l depends on h_l which ultimately depends on the simulated trajectories. Instead, by using independent trajectories we achieve bias-freedom in a simple way.

sSSA defines the next level so that having about $k = N_1 \cdot p$ trajectories passes this level, where p is a non-negligible and non-overwhelming fixed probability. More precisely, we obtain N_1 trajectories by prolonging the successful trajectories in last level until the stopping time. These trajectories are then ordered according to the highest value reached by the level function $h(-)$ in each trajectory. That is, we sort them according to maximum level reaching by a trajectories until the stopping time. Finally, we consider $k = N_1 \cdot p$ and take the $(k + 1)$ topmost trace, and let the next level be h_l.

Having defined the next level E_l, N_2 trajectories are then generated to estimate the probability p_l. The trajectories reaching level E_l are kept for the next stage, while those failing to reach that level are discarded. Let k_l be the actual number of trajectories which hits the level E_l. So, the reaching probability is the ratio of the number of successful traces k_1 over N_2, i.e. $p_l = k_l/N_2$.

For each retained trajectory, we store the first entrance state falling into E_l. The N trajectories in the next stage are simulated starting from these stored states. More in detail, N initial states are picked randomly from the stored ones, possibly duplicating them.

6.3.2 Choosing a level function

The choice of h is a critical issue, a poor selection can lead to a severe inefficiency, even compared to standard methods e.g, SSA. Also, a wrong h which invalidates the assumption $E_l \supseteq E_{l-1}$ can undermine the correctness of our approach. Therefore, we want to discuss on the choosing of level function before going to the details of sSSA in the next section.

The basis of the level function h is to map a multi-dimensional state $X(t)$ into a value representing the importance of that state. It must be consistent with transition paths to drive the system towards the rare event of interest. In other words, it measures how close a given state is from the target rare event E. Function h should return a higher value when the rare event E is more likely to be reached in the next stages. A choice for h in some problems could be easy to define. For example, let consider the event E which is expressed by the number of translocated polymers moves through a narrow pore in the polymer translocation problem in many biological and biotechnological phenomena. Level function h thus could be defined as the number of involving molecules.

A guidance to minimize the variance of the estimator, in general, is to select h so that the probability of reaching an onward level does not depend on the possible entrance states on the trajectory reaching that level (see more details in [162]). In sense of biochemical reactions, the level function should be chosen matching the parts of the reaction mechanism which can increase the probability to reach the event E.

For certain classes of reaction networks which describe a “monotonic” system, structural analysis could help in better understanding the dynamics of the system. The properties of graphical conditions of these reactions will then provide information to the choice of reaction coordinates [8]. However, the biochemical systems are very complex in general which involve many molecular species, interacting through a very complex and nonlinear manner to exhibit consistent and reliable behavior. The qualitative analysis may not be sufficient to account inherently stochastic of these systems, a quantitative method in these cases is preferred. For a complex reaction coordinate, we could exploit the approach presented in [164]. First, the configuration in the transition path is partitioning into set of Voronoi polyhedra. Then the interface is defined as the planes in phase space across the edges of the Voronoi polyhedra.

The committor function, which is the probability a trajectory starting from an intermediate state x will reach the event E started from the initial state x_0, is a precise concept of reaction coordinate [112, 165]. It is the optimal choice of reaction coordinate since it correlates to the progress of the transition paths. Because the committor considers all the the coordinates of the systems, it is a very complex function. We therefore have to project it onto a small set of appropriated coordinates. Then, we have to choose the parameters that most closely matches the committor function which should lead to the interfaces that are perpendicular to the transition paths.

In the above discussion on the choice of the level function, we only focus on reducing the variance of the estimator. However, it is important to note this may not optimize the computation time for the simulation because the cost to reach event E may depend on the entrance states. Further investigating how to address that issue as well is for the time being left for future research.

6.3.3 The sSSA algorithm

We now present our sSSA algorithm, assuming we are given a the level function h as discussed previously. The sSSA procedure is outlined in Alg. 7.

Roughly, in its first main loop iteration sSSA samples an ensemble of N trajectories, which is composed of two groups of N_1 and N_2 trajectories, respectively. We first generate only the first group (N_1 trajectories) (starting from the initial state x_0), which we use to define the next level h_1. We define h_1 so that exactly k trajectories of the generated N_1 reach values of $h(-)$ beyond h_1.

Algorithm 7 sSSA procedure

```
require: $N = N_1 + N_2$: total number of trajectories
 1:         $k$: expected number of successful trajectories at each level
 2:         $k_s$: number of successful trajectories to stop

 3: set $l = 1$
 4: set $X^i$ to be a single-point trajectory starting from state $x_0$ at time $0$, for each $i = 1..N$

    main loop:
 5: repeat
 6:     prolong the $N_1$ trajectories $X^i$ with $i = 1..N_1$ using SSA from their last state until
    stopping time $T_{max}$
 7:     compute maximum level $h^i$ reached by $X^i$ by $h^i = \max_{0 \leq t \leq T_{max}} h(X^i(t))$
 8:     set $h_l = (k+1)$th quantile of $\{h^i\}_{i=1}^{N_1}$
 9:     prolong the $N_2$ trajectories $X^{N_1+j}$ with $j = 1..N_2$ using SSA from their last state until
    stopping time $T_{max}$
10:     let $k'$ be the number of trajectories in the $N_2$ group reaching the target rare event
11:     within the $N_2$ group, retain the $k_l$ trajectories reaching level $h_l$, and truncate them at the
    first time they do so
12:     compute $p_l = k_l / N_2$
13:     uniformly clone to obtain $N$ trajectories $\{X^i\}_{i=1}^{N}$ within the $k_l$ retained ones, update the
    old values for $X^i$
14:     increase $l = l + 1$
15: until $k' \geq k_s$
16: let $p_l = k' / N_2$

    statistics:
17: produce estimated probability $\tilde{\gamma} = \prod_{i=1}^{l} p_i$
```

Formally, let h^i, be the maximum value obtained by evaluating function h on corresponding trajectory X^i, as shown below:

$$h^i = \max_{0 \leq t \leq T_{max}} h(X^i(t)) \tag{6.5}$$

The next level h_1 is then chosen to be the $(k+1)$th quantile in these N_1 values. Hence, h_1 is the $(k+1)$th quantile of $\{h^i\}_{i=1}^{N_1}$. By doing this, we make the probability of reaching the next level E_1 starting from $x_0 \in E_0$ close to k/N_1. Parameters k and N_1 can be tuned as needed, so to make this probability non negligible.

Then, we sample other N_2 trajectories (from x_0), and count how many of them actually reach the next level E_1. We let p_1 to be the ratio of the number of successful traces k_1 over N_2, which we use as an estimator of $\mathbb{P}(E_1|E_0)$.

We prepare for the next iterations by uniformly sampling N elements from the set of *successful* trajectories in the N_2 group. (Note that since $N > N_2$, some trajectories will be taken more than once.) Name these trajectories $\{X^i\}_{i=1..N}$. Then, we truncate each X^i at the time it first succeeds, i.e. at its first crossing of the next level. The next iterations can then start. The sSSA algorithm will repeat the tasks done in the first iteration, except for the fact that the new $N = N_1 + N_2$ generated trajectories are not simulated starting from x_0 but rather from the last states of the retained X^i trajectories. In other words, we prolong each X^i until time T_{max}. In this way, we start the new simulations from states in E_l (the last states of X^i), so that the computation of $p_l = k_l/N_2$ indeed estimates $\mathbb{P}(E_{l+1}|E_l)$.

We stop the main loop when a significant part (at least k_s) of the generated N_2 trajectories hit the target rare event. When that happens, we just estimate the probability of the rare event by the chain rule $\hat{\gamma} = \prod_{i=1}^{l} p_i$, where l is the number of levels which have been generated.

The estimator $\hat{\gamma}$ is clearly bias-free, but also does not required *a priori* in determining the levels. The asymptotic normality convergent of the estimator could be proved using the formulation in [31]. While setting the expected probability for reaching the next level, we have tried to reduce the variance of the estimator. In the special case where $\gamma = p^q$ where $q \in \mathbb{N}$, with the choice of the levels such that all $p_l = p$ then it is the same as the optimal setting in fixed multilevel splitting (see [67] for more detailed discussion on the optimal conditions).

Compared with the standard sampling using SSA, our algorithm requires a little more computational resources given the same model and parameters. A rough analysis of sSSA can be done as follows. Assume T_{max} to be a constant, and let C be the expected work to generate a single trajectory. By applying e.g. tree search, the search and update in m reactions can be done in logarithmic time in the number of reactions, i.e., $C = O(\log m)$. The expected work for generating N trajectories is thus $O(NC) = O(N \log m)$. To select the $(k+1)$th quantile in N_1 values a direct selection search or a randomized version [38] which run in expected linear time, i.e., $O(N_1)$ could be applied. Overall, the expected work is therefore $O(N \log m + N_1)$ for sSSA, which is only slightly more than SSA. This comparison however does not take accuracy into account, which is crucial in the problem at hand. To better compare SSA and sSSA we resort to experiments in the next section.

6.4 Experimental results

In this section we report on the experimental results of our algorithm on two models: a simple production degradation model and an artificial biological switch model. With these model we compare the performance and efficiency of sSSA with standard simulation methods. The code was implemented and run on the Intel Core i5-540M processor.

Table 6.1: Production degradation model

R_1: $DNA \to DNA + mRNA$	$k_1 = 1$
R_2: $mRNA \to \emptyset$	$k_2 = 0.025$

6.4.1 Production degradation model

The production degradation model consists of two species DNA, $mRNA$ and involves two reactions shown in Table 6.1. The system models a simple transcription of DNA to $mRNA$. In this model, the DNA continuously produces $mRNA$ via reaction R_1 at rate k_1, while $mRNA$ is degraded at rate k_2 in reaction R_2. The initial state $X(0)$ with each component is given with $\#DNA(0) = 1$ and $\#mRNA(0) = 40$.

In this example we aim to estimate the probability that the first time the system reaches a state in which the population of $mRNA$ is larger than a threshold value λ, which is taken in the range $55, 60, 65, 70,$ and 75 respectively. The event is meant to be reached before the stopping time $T_{max} = 100$ and given the initial state $X(0)$. First, we briefly derive the complete time evolution of the system following CME in this simple case, and calculate the probability the first time the system reaches the event. Then we compare with the simulation methods, i.e., SSA and sSSA.

To do so, let a_1 and a_2 be the propensities of reaction R_1 and R_2, respectively. We have:

$$a_1(X) = k_1\#DNA \tag{6.6}$$

$$a_2(X) = k_2\#mRNA \tag{6.7}$$

$$a_0(X) = a_1(X) + a_2(X) \tag{6.8}$$

Notice that in our model the population of DNA is conserved, so the a_1 is unchanged overtime, and thus we only focus on the change of species $mRNA$. The state vector now is reduced to one dimensional $X = \#mRNA$.

Let $P(\lambda, t|\lambda_0, 0)$ be the probability density that t is the first time in which $\#mRNA(t) \geq \lambda$, given initial value $\#mRNA = \lambda_0$. We have the CME equation written as

$$\frac{\delta}{\delta t}P(\lambda, t|\lambda_0, 0) = a_1(\lambda - 1)P(\lambda - 1, t|\lambda_0, 0) + \\ a_2(\lambda + 1)P(\lambda + 1, t|\lambda_0, 0) - a_0(\lambda)P(\lambda, t|\lambda_0, 0) \qquad (6.9)$$

The collection of differential-difference equations in Eq. 6.9 is analytically tractable, and can be solved given the initial constraints $P(\lambda_0, 0|\lambda_0, 0) = 1$ and $P(-1, t|0, 0)) = P(\lambda + 1, t|\lambda_0, 0) = 0$. The propensities are also confined to zero at state λ since the system stays in state λ forever upon the first arrival.

Let expand Eq. 6.9 using a matrix representation by defining matrix M having size $(\lambda + 1) \times (\lambda + 1)$ where $M[i, i] = a_0(i - 1)$, $M[i + 1, i] = -a_2(i)$ and $M[i, i + 1] = -a_1(i - 1)$, while all other elements including the last column are set to zero. The notation $M[m, n]$ denotes the element at row m and column n of matrix M. Let $Q(t)$ be the probability vector of all probablely reachable states at time time t, i.e., $Q(t) = (P(0, t|\lambda_0, 0), ..., P(\lambda, t|\lambda_0, 0))^T$. The Eq. 6.9 is then rewritten as

$$\frac{\delta}{\delta t}Q(t) = -MQ(t) \qquad (6.10)$$

Denote $\mu_1 < \mu_2 < ... < \mu_{\lambda+1}$ be the eigenvalues of matrix M in increasing order. Let $e^{-\mu t}$ be the diagonal matrix with value $e^{\mu_i t}$ in the main diagonal, and V be the matrix that its ith column is the eigenvector of M corresponding to eigenvalue μ_i, respectively. The solution of Eq. 6.10 has the following form

$$Q(t) = Ve^{-\mu t}V^{-1}Q(0) \qquad (6.11)$$

By expanding it, we receive $P(\lambda, t|\lambda_0, 0)$ as

$$P(\lambda, t|\lambda_0, 0) = \sum_{i=1}^{\lambda+1} V[\lambda + 1, i]e^{-\mu_i t}V^{-1}[i, \lambda_0 + 1] \qquad (6.12)$$

From these results we can conclude that if the time $t = T_{max}$ is fixed, then the probability $\#mRNA$ reaching λ becomes very small as increasing λ. For example, with our setting above taking $\lambda = 75$, we have $\gamma = 8.4171 * 10^{-6}$. In other words, the probability to reach the event is indeed rare.

To compare the result of SSA and sSSA, we run them, in turn, with different values of N, in which 20% are use to determine levels, taken from $10^3, 10^4$ and 10^5 for each value of λ. For choosing level we expected 50% of the trajectories will pass to next level. The simulation stop when having at least 30% of trajectories hit the event. To estimate the variance of the estimator by sSSA we run the simulation 100 times. For the level function h we choose it to be the number of mRNA. It is clear since the model has been reduced to a one dimensional system. The results of the experiments to estimate γ are shown in Table 6.2 for different values of λ. Table 6.2 gives the estimated probabilities, with the estimated relative error for different settings of SSA and sSSA. Note that we write '-' meaning that there are no successful trajectories hitting the event E. When the probability is not rare, say from 10^{-1} to 10^{-3}, SSA can be used to roughly estimate the probability the first time the system reaching the event, as in the cases $\lambda = 55, 60, 65$. when the probability instead becomes rarer and we still use a fixed budget N, SSA is not so accurate. For example, in case $\lambda = 75$ there is no successful trajectory reaching the event. Hence, we could not approximate $\hat{\gamma}$. In this extreme case, sSSA still could be able to estimate $\tilde{\gamma}$ even with only $N = 10^3$ trajectories.

Furthermore, we can conclude from the Table 6.2 the estimated variance by sSSA in all cases is always better than SSA with the same parameters setting. The estimated RE of algorithms while λ takes small values, in corresponding to not rare event, is not too much, although sSSA is always smaller. An interesting point from results in Table 6.2, even though in the case of not rare, is that sSSA could produce the same RE by using less simulation time than SSA. For example, to have $RE = 2\%$ in case $\lambda = 60$, SSA requires 10^5 trajectories with simulation time 16 seconds while sSSA just needs 5 seconds with $N = 10^4$. Similarly, for the case $\lambda = 65$, sSSA just requires $N = 10^3$ with time less than

Table 6.2: Estimated probability for Production degradation model of simulation methods with initial state $\#DNA = 1, \#mRNA = 40$. '-' means there are no successful trajectories hitting the event

λ	Analytic	Simulation						
		N	SSA			sSSA		
			Prob.	RE	Time(ms)	Prob.	RE	Time(ms)
55	0.1186	10^3	0.112	8.41E-02	220	0.1046	5.60E-02	397
		10^4	0.1167	2.71E-02	1644	0.1105	1.51E-02	3375
		10^5	0.1193	8.64E-03	15261	0.1107	3.51E-03	33108
60	0.0207	10^3	0.018	2.03E-01	262	0.0164	6.76E-02	580
		10^4	0.0177	6.37E-02	1701	0.0173	2.21E-02	5368
		10^5	0.0201	2.14E-02	15844	0.0203	9.47E-03	35018
65	0.0023	10^3	0.004	8.68E-01	279	0.0023	6.97E-02	884
		10^4	0.0018	1.84E-01	1751	0.0019	2.57E-02	7961
		10^5	0.0021	6.26E-02	16353	0.0021	8.65E-03	52895
70	1.68E-4	10^3	-	-	-	2.18E-4	5.23E-02	1540
		10^4	1.00E-4	5.97E-01	1856	1.68E-4	2.72E-02	10458
		10^5	1.30E-4	2.15E-01	17368	1.66E-4	6.31E-03	354895
75	8.42E-6	10^3	-	-	-	5.14E-06	5.73E-02	8760
		10^4	-	-	-	7.29E-06	3.69E-02	33705
		10^5	-	-	-	8.57E-06	7.69E-03	437547

second to achieve $RE = 6\%$ while SSA needs to run 16 seconds with $N = 10^5$ trajectories.

The difference in RE between algorithms becomes very large when the probability became very small. The number of trajectories used by SSA has to grow linear with the rarity. This mean the RE of SSA will very poor if we fix N and increase the rarity. This is exactly what we obtained from the Table 6.2. But, sSSA instead scales very well. The RE in estimating the event is controlled around 5% in case we only use $N = 10^3$. In other words, the convergence of the sSSA estimator is really better than the standard method.

We also study the computational cost of sSSA. Given a fixed N, using the algorithms discussed above, the runtime of sSSA is always a bit longer than SSA since it has to resample to obtain more successful trajectories. However, to measure the efficiency of the estimator in the rare event setting, while also taking into account the computational cost, we study the *efficiency* of an estimator, which is given by the inverse of the multiplication of the variance and the expected runtime to estimate γ, i.e., Eff($\hat{\gamma}$) = $1/[Var(\hat{\gamma}) * T(\hat{\gamma})]$, where $Var(\hat{\gamma})$ is the estimated variance and $T(\hat{\gamma})$ denote the simulation time. Thus, an estimator $\hat{\gamma}$ is said efficient than $\tilde{\gamma}$ if it has greater efficiency, i.e., when Eff($\hat{\gamma}$) $>$ Eff($\tilde{\gamma}$). The result is presented in Fig. 6.3 showing the efficiency of SSA and sSSA.

Although sSSA demands more CPU runtime than SSA, given in Table 6.2, its efficiency is better than SSA in all cases as shown in Table 6.3 where we compare with each value of N. Note that, however, when the event is not rare, for example in case $\lambda = 55$, the additional computational expensive will downward the efficiency, resulting in a suboptimal efficiency of sSSA. For example, the case $N = 10^4$ and $\lambda = 55$, we have Eff(sSSA) $= 0.4470$ and Eff(SSA) $= 0.4556$, showing that SSA is a little better, nearly one percentage, than sSSA. By contrast, when $\lambda = 75$ and $N = 10^4$, sSSA is much more efficient, since Eff(sSSA) $= 813.5235$ while there are no successful trajectories in case of SSA

Table 6.3: Efficiency of simulation methods for estimating probabilities the first time the population of $\#mRNA$ reaching λ for Production Degradation model with initial state $\#DNA = 1, \#mRNA = 40$

N	λ	Eff(SSA)	Eff(sSSA)
10^3	55	0.4556	0.4470
	60	0.9074	1.7233
	65	1.7948	10.1436
	70	0	56.9536
	75	0	387.5949
10^4	55	0.1894	0.1985
	60	0.4458	0.4872
	65	1.3472	2.5724
	70	5.3879	20.9253
	75	0	110.2641
10^5	55	0.0641	0.0883
	60	0.1424	0.2174
	65	0.4244	1.0407
	70	1.5969	2.6901
	75	0	34.6791

Table 6.4: Biological switch model

No.	Reaction	Rate constant
R_1	$2A \to A_2$	$k_1 = 5$
R_2	$A_2 \to 2A$	$k_2 = 5$
R_3	$O + A_2 \to OA_2$	$k_3 = 5$
R_4	$OA_2 \to O + A_2$	$k_4 = 1$
R_5	$O \to O + A$	$k_5 = 1$
R_6	$OA_2 \to OA_2 + A$	$k_6 = 1$
R_7	$A \to \emptyset$	$k_7 = 0.25$
R_8	$2B \to B_2$	$k_8 = 5$
R_9	$B_2 \to 2B$	$k_9 = 5$
R_{10}	$O + B_2 \to OB_2$	$k_{10} = 5$
R_{11}	$OB_2 \to O + B_2$	$k_{11} = 1$
R_{12}	$O \to O + B$	$k_{12} = 1$
R_{13}	$OB_2 \to OB_2 + B$	$k_{13} = 1$
R_{14}	$B \to \emptyset$	$k_{14} = 0.25$

(in those cases we set Eff(SSA) $= 0$).

Suppose we want to use SSA for estimating the probability the first time the population of $\#mRNA$ reaching $\lambda = 75$ with $RE = 5\%$. We therefore have to simulate roughly $1/(RE * \gamma) \approx 10^7$ trajectories. The average time for one trajectory generating is 12ms in our machine. In case we use sSSA with total $N = 10^3$, the computational gain returning by our algorithm is $(12 * 10^7)/(8760) \approx 10^4$.

6.4.2 Biological switch model

In this case study, we applied the sSSA to a generic biochemical switch [170]. It is an artificial model of a minimal presentation of lysis/lysogeny switch in the phage λ which consists of two adjacent operons that mutually repress each other. There is a demonstrated construction of the toggle switch in E. coli proposed in [56]. The reaction model is shown in 6.4. We consider here the exclusive model of toggle switch in which only one dimer can bind to the DNA. In particular, the generic switch consists of two factors: proteins A and protein B which is encoding by their corresponding genes $\mathcal{A}$ and $\mathcal{B}$. Protein A and B form the corresponding homodimers A_2 and B_2, respectively, which can bind to DNA, named O. When A_2 is bound, gene $\mathcal{B}$ is not transcribed, and in vice versa B_2 is bound, it suppresses the transcription of gene $\mathcal{A}$. As the DNA is bound by one protein type, it continuously produce the corresponding protein. Thus this biological switch model shows the appearance of bistability phenomenon where there are two steady states corresponding with the high number of protein A and B, respectively.

Let $N_A = \#A + 2(\#A_2 + \#OA_2)$ and $N_B = \#B + 2(\#B_2 + \#OB_2)$ be the total number of protein A and B. In this model we focus on estimating the probability of transition starting from a state in the region with $N_A = \lambda_A$ and ending in another region with $N_B \geq \lambda_B$ during the simulation time T_{max}. Because of the mutual suppression between protein A and B, the probability to move from one highly stable region to the opposite stable region will becomes very rare. Table 6.5 shows the estimated probability for our case study with different settings for initial values of λ_A, while the total number of protein B is 0.

From the result, we draw a conclusion that the estimated probability by sSSA is consistent with the SSA when the event is less rare, the case $\lambda_A = 20$. While increasing λ_A, the event becomes indeed rare. In other words the probability to jump off this region is very small. The number of simulation N of SSA has to grow proportional with the rarity of the event. For example, $\lambda_A = 40$, the order of probability is 10^{-8}, we have to generate in average $N = 10^{12}$ trajectories to estimate the probability with $RE = 1\%$. The average time to have one

Table 6.5: Estimated probability for Biological Switch model of simulation methods with fixed $\lambda_B = 25$, while changing initial value of λ_A. '-' means there are no successful trajectories hitting the event

λ_A	SSA				sSSA			
	N	Prob.	RE	Time(min)	N	Prob.	RE	Time(min)
20	10^5	5.20E-04	1.35E-01	128	10^4	4.70E-04	5.21E-02	97
	10^6	4.95E-04	5.16E-02	1299	10^5	4.54E-04	2.29E-02	964
	10^7	4.63E-04	1.70E-02	11633				
30	10^5	-	-	-	10^4	2.42E-6	5.68E-02	218
	10^6	-	-	-	10^5	1.34E-6	2.36E-02	2015
	10^7	1.20E-6	2.89E-01	18874				
40	10^5	-	-	-	10^4	4.23E-8	6.10E-02	341
	10^6	-	-	-	10^5	2.14E-8	2.79E-02	3341
	10^7	-	-	-				

trajectory for this model is 235ms. The simulation is therefore unaceptable (roughly 7451 years!). While the sSSA could estimate this probablity with total $N = 10^4$ trajectories with controlled 6% relative error.

6.5 Conclusions

Stochastic simulation is an emerging research area for investigating biological processes, especially whenever fluctuation and noise play important roles. Living organisms use different mechanisms, which usually involve the complex and nonlinear interactions between molecular species, to expose a consistent behavior under such noisy regimes. Hence, rare event simulation becomes a very important step to understand the robustness and the reliability of biochemical systems. In this paper we developed a new algorithm, called sSSA, to improve the efficiency w.r.t. standard stochastic simulation in a rare event setting.

Although the sSSA algorithm has been shown to be efficient when applied to a few reference models, further investigation is necessary. A first line of research would be to find more guidelines helping a modeler to choose the of level function h. This is important since a bad choice for h can lead to a lower efficiency, even when comparing with previous methods. Second, from the practical point of the algorithm, we need to decide the number of levels in the level sequence to achieve a better performance in applying to simulate a real biochemical systems. In this work we prolong a trajectory from the first entrance state when it reaches the next level. We clearly can extend this to count all the states falling in the next level. And a pruning technique, e.g., Russian roulette, could be applied to kill trajectories going down to save the computational resources. A further study on the efficiency of the algorithm is required.

Chapter 7

Conclusion

Stochastic simulation is an invaluable tool for understanding the complexities of biochemical reactions. In this thesis we studied performance of the exact stochastic simulation algorithm i.e., SSA, and contribute to the development of new efficient formulations. We proposed new algorithms for improving both efficiency and statistical accuracy measurement of the stochastic simulation so to make it applicable for large and highly coupled reaction networks.

In chapter 3 we study the effect of the search of next reaction firing to the performance of the stochastic simulation. We proposed a tree-based search approach to reduce the search time complexity. Through the experiments, we showed that simulation performance can be sensibly improved if an underlying tree data structure is used to support the search. We predict the shape of the tree leading to optimal average search time. This turns out to be the Huffman tree, a device used in computer science for data compression. Then, we study the impact of approach to rebuild the tree when it becomes non-optimal by many reaction firings. Two approaches are presented to handle this problem namely: the fixed time tree rebuilding and adaptive time tree rebuilding in which the latter allows to rebuilt the tree during the simulation depending on how the system evolves.

Then, we study the effect of the propensity update to overall performance of stochastic simulation. Indeed, whenever the population of a species is changed by a reaction firing, the propensities of all the dependent reactions has to be recomputed. Even though a dependency graph can reduce the update to be model-dependent, in which only locally affected reactions have to recompute their propensities, still there are many models, e.g., highly coupled reactions, where a costly update required. A significant portion of the computation time is spent on propensity updates. In highly coupled reaction networks, the propensity updates soon become a bottleneck of the whole algorithm. In chapter 4 we proposed an solution to cope this problem with a new simulation algorithm, called *RSSA*. RSSA is an exact simulation algorithm improving the simulation performance by postponing and collapsing as much as possible the propensity updates, hence reducing their cost. RSSA exploits a rejection-based mechanism to select a reaction firing. It uses the over approximation reaction propensity, which is often very fast and more efficient to compute, to select a reaction firing. The search of a reaction firing is carried out in two steps. A candidate reaction is selected according to an over-approximation of its propensity. A rejection step is then applied to recover the exactness of the algorithm. We further contribute to the improvement of RSSA in both of these steps. First, we discuss which search procedures for finding a candidate reaction lead to better performances, for different network sizes. Second, we study several strategies for controlling the amount of over-approximation, and analyze their impact to the simulation performance. We also discuss how to systematically optimize the tunable parameters of RSSA so to maximize its performance.

In chapter 5 we extend our study to the reaction networks where the diffusion significantly affects the biological behaviour. The spatial extensions of SSA are introduced to simulate the reaction-diffusion by dividing the space into subvolumes so that a subvolume is assumed to be well-mixed. The diffusion in this modelling is explicitly modelled as unimolecular reaction. The search of a reaction firing now is consisting of two steps: 1) search for a subvolume, and then 2) search for a reaction firing in that subvolume. After a selected reaction fires the system is updated. Although a spatial SSA is able to simulate the reaction-diffusion processes, its performance is slow due to the inefficient search and update. In chapter 5 we propose a new formulation, called RRD. Our new formulation combines an efficient binary search and approximation of propensity for searching an subvolume and then a reaction firing inside that subvolume. According to our experiments, the search and update of a reaction firing in a subvolume by our formulation is substantially reduced and thus its performance outperforms over previous approaches e.g., NSM.

In chapter 6 we study the statistical analysis of targeted event of interest by performing stochastic simulation. The random in reaction firing requires a large number of simulation runs to achieve a reasonable statistical accuracy. The task becomes increasingly harder when considering *rare events,* which occur only with a very small probability. Estimating the probability of rare events in biochemical systems, however, is an important task, since it can help in studying rare abnormal behavior when they do occur. We contribute to this study by proposing a new algorithm, called *sSSA*, to efficiently estimate the probability of a rare event. It is a kind of biased simulation where the state space is *split* into subsets so that the event become more likely when moving from one subset to another. Thus, simulated trajectories are gradually "pushed" towards the rare event following such subsets. The (unbiased) probability for the rare event is then estimated by counting the successful (biased) trajectories, and then applying a correction factor so to account for the bias.

Concluding, we investigated new algorithms for improving exact stochastic simulation; however, there still many problems are open for further investigation. For instance, in exact stochastic simulation a discrete copy number of each molecular species is keep tracking. This is obviously not efficient because the population of species in biological systems is often great disparity. The species having large population should be better simulated by a fast, but less sensitive, simulation algorithm without loss of accuracy. A hybrid simulation algorithm is an ideal approach to handle this problem in which the large population species is handled by a fast numerical integration, e.g., ODE integration, while the low population species is simulated by an exact simulation algorithm. RSSA is a good candidate for the exact simulation algorithm. In hybrid approach, the deterministic integration part can affect the stochastic part. However, by RSSA, we do not need to update the propensity of affected reactions in stochastic part if the species population is still confined in its fluctuation interval. Thus, a lot of computation effort would be saved. The parallel stochastic simulation is also a promising approach to deal with the complexity of biochemical systems. However, because the stochastic simulation is inherently sequential it requires special approach to parallelize the algorithm to achieve a better performance. Finally, extending stochastic simulation for systems that are not well-mixed also exposes a great many challenges.

Bibliography

[1] Luca Aceto, Anna Ingolfsdottir, Kim G. Larsen, and Jiri Srba. *Reactive Systems: Modelling, Specification and Verification*. Cambridge University Press, 2007.

[2] Aurlien Alfonsi, Eric Cancs, Gabriel Turinici, Barbara Di Ventura, and Wilhelm Huisinga. Adaptive simulation of hybrid stochastic and deterministic models for biochemical systems. In *Proc. of ESAIM*, 2005.

[3] Michael P. Allen. Introduction to molecular dynamics simulation. In *Lecture Notes on Computational Soft Matter: From Synthetic Polymers to Proteins*, volume 23, pages 1–28, 2004.

[4] R. Allen, C. Valeriani, and P. ten Wolde. Forward flux sampling for rare event simulations. *J. Phys.*, 21:463102, 2009.

[5] S. Alonso, F. Sagus, and A. S. Mikhailov. Negative-tension instability of scroll waves and winfree turbulence in the oregonator model. *J. Phys. Chem. A*, 110(43):12063–12071, 2006.

[6] David F. Anderson. A modified next reaction method for simulating chemical systems with time dependent propensities and delays. *J. Chem. Phys.*, 127(21):214107, 2007.

[7] Steven S Andrews and Dennis Bray. The two-regime method for optimizing stochastic reaction-diffusion simulations. *Phys. Biol.*, 1:137–151, 2004.

[8] D. Angeli, P. De Leenheer, and E. Sontag. Graph-theoretic characterizations of monotonicity of chemical networks in reaction coordinates. *J Math Biol.*, 61(4):581–616, 2010.

[9] Adam Arkin, John Ross, and Harley H. McAdams. Stochastic kinetic analysis of developmental pathway bifurcation in phage lambda-infected escherichia coli cells. *Genetics*, 149:16331648, 1998.

[10] Uri M. Ascher and Linda R. Petzold. *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations*. SIAM, 1998.

[11] A Auger, P Chatelain, and P Koumoutsakos. R-leaping: accelerating the stochastic simulation algorithm by reaction leaps. *J. Phys. Chem.*, 125(8):084103, 2006.

[12] F. Baras and M. Malek Mansour. Reaction-diffusion master equation: A comparison with microscopic simulations. *Physical Review E*, 54(6), 1996.

[13] M. Barrio, K. Burrage, A. Leier, and T. Tian. Oscillatory regulation of hes1: Discrete stochastic delay modelling and simulation. *PLoS Comput. Biol.*, 2(9):117, 2006.

[14] O. Bastiansen. *The Law of Mass Action, A Centenary Volume*. Universitetsforlaget: Oslo, 1964.

[15] Basil Bayati, Philippe Chatelain, and Petros Koumoutsakos. Adaptive mesh refinement for stochastic reactiondiffusion processes. *Journal of Computational Physics*, 230:13–26, 2011.

[16] M Bentele and R. Eils. General stochastic hybrid method for the simulation of chemical reaction processes in cells. In *Proc. of Computational Methods in Systems Biology (CMSB)*, 2004.

[17] David Bernstein. Simulating mesoscopic reaction-diffusion systems using the gillespie algorithm. *Phys. Rev. E*, 71(4):041103, 2005.

[18] James Blue, Isabel Beichl, and Francis Sullivan. Faster monte carlo simulations. *Phys. Rev. E*, 51(2):867–868, 1995.

[19] D. Bratsun, D. Volfson, Lev S. Tsimring, and J. Hasty. Delay-induced stochastic oscillations in gene regulation. In *In Proc. of PNAS*, 2005.

[20] R. Brown. Calendar queues: a fast 0(1) priority queue implementation for the simulation event set problem. *Communications of the ACM*, 31(10):1220–1227, 1988.

[21] X Cai. Exact stochastic simulation of coupled chemical reactions with delays. *J. Chem. Phys.*, 126(12):124108, 2007.

[22] X Cai and Z Xu. K-leap method for accelerating stochastic simulation of coupled chemical reactions. *J. Phys. Chem.*, 126:074102, 2007.

[23] Y Cao, D Gillespie, and L Petzold. Multiscale stochastic simulation algorithm with stochastic partial equlibrium assumption for chemically reacting systems. *J Comp. Phys.*, 266(2):395411, 2005.

[24] Y Cao, D Gillespie, and L Petzold. The slow-scale stochastic simulation algorithm. *J. Chem. Phys.*, 122(1):014116, 2005.

[25] Y Cao, D T. Gillespie, and L R. Petzold. Avoiding negative populations in explicit poisson tau-leaping. *J. Phys. Chem.*, 123(5):054104, 2005.

[26] Y Cao, D T. Gillespie, and L R. Petzold. Trapezoidal tau-leaping formula for the stochastic simulation of biochemical systems. In *Proc. of Foundations of Systems Biology Engineering (FOSBE 2005)*, 2005.

[27] Y Cao, D T. Gillespie, and L R. Petzold. Efficient step size selection for the tau-leaping simulation method. *J. Phys. Chem.*, 124:044109, 2006.

[28] Y Cao, D T. Gillespie, and L R. Petzold. Adaptive explicit-implicit tau-leaping method with automatic tau selection. *J. Phys. Chem.*, 126(22):224101, 2007.

[29] Yang Cao, Hong Li, and Linda Petzold. Efficient formulation of the stochastic simulation algorithm for chemically reacting systems. *J. Chem. Phys.*, 121(9):4059, 2004.

[30] F. Cerou and A. Guyader. Adaptive multilevel for rare event analysis. *Stochastic Analysis and Applications*, 25(2):417–443, 2007.

[31] F. Cerou, P. Moral, F. LeGland, and P. Lezaud. Genetic genealogical models in rare event analysis. *ALEA Lat. Am. J. Prob. Math. Stat.*, 1:181203, 2006.

[32] A. Chatterjee, D. G. Vlachos, and M. A. Katsoulakis. Binomial distribution based tau-leap accelerated stochastic simulation. *J. Phys. Chem.*, 122(2):024112, 2005.

[33] N A. Chebotareva, B I. Kurganov, and N B. Livanova. Biochemical effects of molecular crowding. *Biochemistry*, 69(11):1239–1251, 2004.

[34] William W. Chen, Birgit Schoeber, Paul J. Jasper, Mario Niepel, Ulrik B. Nielsen, Douglas A. Lauffenburger, and Peter K. Sorger. Input-output behavior of ErbB signaling pathways as revealed by a mass action model trained against dynamic data. *Mol. Syst. Biol.*, 5(239), 2009.

[35] TaiJung Choi, Mano Ram Maurya, Daniel M. Tartakovsky, and Shankar Subramaniam. Stochastic operator-splitting method for reaction-diffusion systems. *J. Chem. Phys.*, 137:184102, 2012.

[36] B. Chopard and M. Droz. *Cellular Automata Modeling of Physical Systems*. Cambridge University Press, 1998.

[37] Anne Condon, David Harel, Joost N. Kok, Arto Salomaa, and Erik Winfree. *Algorithmic Bioprocesses*. Springer, 2009.

[38] T. Cormen, C. Stein, R. Rivest, and C. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2001.

[39] E. J. Crampina, S. Schnella, and P. E. McSharry. Mathematical and computational techniques to deduce complex biochemical reaction mechanisms. *Progress in Biophysics and Molecular Biology*, 86(1):77–112, 2004.

[40] Vincent Danos, Jerome Feret, Walter Fontana, and Jean Krivine. Scalable simulation of cellular signaling networks. In *Proc. of APLAS*, 2007.

[41] R. Das, V. Esposito, M. Abu-Abed, G. Anand, S. Taylor, and G. Melacini. camp activation of pka defines an ancient signaling mechanism. In *Proc. of PNAS*, 2006.

[42] Gerda de Vries, Thomas Hillen, Mark Lewis, Birgitt Schnfisch, and Johannes Muller. *A Course in Mathematical Biology: Quantitative Modeling with Mathematical and Computational*. SIAM, 2006.

[43] Luc Devroye. *Non-Uniform Random Variate Generation*. Springer-Verlag, 1986.

[44] A R. Dinasarapu, B. Saunders, I Ozerlat, K. Azam K, and S. Subramaniam. Signaling gateway molecule pages - a data model perspective. *Bioinformatics*, 27(12):17361738, 2010.

[45] Johan Elf, Andreas Doni, and Mns Ehrenberg. Mesoscopic reaction-diffusion in intracellular signaling. In *In Proc. of SPIE*, 2003.

[46] M. Elowitz, A. Levine, E. Siggia, and P. Swain. Stochastic gene expression in a single cell. *Science*, 297:11831186, 2002.

[47] Stefan Engblom, Lars Ferm, Andreas Hellander, and Per Lotstedt. Simulation of stochastic reaction-diffusion processes on unstructured meshes. *SIAM J. Sci. Comput.*, 31(3):1774–1797, 2009.

[48] Radek Erban and S Jonathan Chapman. Stochastic modelling of reactiondiffusion processes: algorithms for bimolecular reactions. *Physical Biology*, 6(4), 2009.

[49] R. Ewald, C. Maus, A. Rolfs, and A. Uhrmacher. Discrete event modelling and simulation in systems biology. *Journal of Simulation*, 1:8196, 2007.

[50] James R. Faeder, Michael L. Blinov, Byron Goldstein, and William S. Hlavacek. Rule-based modeling of biochemical networks. *Complexity*, 10(4):2241, 2005.

[51] David Fange, Otto G. Berg, Paul Sjberg, and Johan Elf. Stochastic reaction-diffusion kinetics in the microscopic limit. In *Proc. of PNAS*, 2010.

[52] David Fange and Johan Elf. Noise-induced min phenotypes in e. coli. *PLoS Comput. Biol.*, 2(6), 2006.

[53] D. Fell. *Understanding the Control of Metabolism*. Portland Press, 1997.

[54] A. Finney and M. Hucka. Systems biology markup language: Level 2 and beyond. *Biochemical Society Transactions*, 31(6):1472, 2003.

[55] M B Flegg, S J Chapman, and R. Erban. The two-regime method for optimizing stochastic reaction-diffusion simulations. *Journal of The Royal Society Interface*, 9(70):869–68, 2012.

[56] T. Gardner, C. Cantor, and J. Collins. Construction of a genetic toggle switch in escherichia coli. *Nature*, 403:339–342, 2000.

[57] M. Garvels. *The splitting method in rare event simulation*. PhD thesis, University of Twente, 2000.

[58] J. Gentle, W. Hrdle, and Y. Mori. *Handbook of Computational Statistics: Concepts and Methods*. Springer-Verlag, 2012.

[59] Michael Gibson and Jehoshua Bruck. Efficient exact stochastic simulation of chemical systems with many species and many channels. *J. Phys. Chem. A*, 104(9):1876–1889, 2000.

[60] Daniel T. Gillespie. A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. *J. Comp. Phys.*, 22(4):403–434, 1976.

[61] Daniel T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *J. Phys. Chem.*, 81(25):2340–2361, 1977.

[62] Daniel T. Gillespie. Approximate accelerated stochastic simulation. *J. Phys. Chem.*, 115(4):171633, 2001.

[63] Daniel T. Gillespie. Approximate accelerated stochastic simulation of chemically reacting. *J. Chem. Phys.*, 115:1716–1733, 2001.

[64] Daniel T. Gillespie. A rigorous derivation of the chemical master equation. *Physica A*, 188(1):404–425, 2007.

[65] Daniel T. Gillespie. Stochastic simulation of chemical kinetics. *Annu Rev Phys Chem.*, 58:35–55, 2007.

[66] Daniel T. Gillespie and Linda R. Petzold. Improved leap-size selection for accelerated stochastic simulation. *J. Phys. Chem.*, 119(16):822934, 2003.

[67] P. Glasserman, P. Heidelberger, P. Shahabuddin, and T. Zajic. Multilevel splitting for estimating rare event probabilities. *Operations Research*, 47(4):585–600, 1999.

[68] R Goh and I. Thng. Mlist: an efficient pending event set structure for discrete event simulation. *Journal of Simulation*, 4(5), 2003.

[69] P. Goss and J. Peccoud. Quantitative modeling of stochastic systems in molecular biology by using stochastic petri nets. In *PNAS*, 1998.

[70] S. Goutelle, M. Maurin, F. Rougier, X. Barbaut, L. Bourguignon, M. Ducher, and P. Maire. The Hill equation: a review of its capabilities in pharmacological modelling. *Fundamental & Clinical Pharmacology*, 22, 2008.

[71] A. Guyader, N. Hengartner, and E. Matzner-Lber. Simulation and estimation of extreme quantiles and extreme probabilities. *Appl. Math. Optim.*, 64(2):171–196, 2011.

[72] E L Haseltine and J B Rawlings. Approximate simulation of coupled fast and slow reactions for stochastic chemical kinetics. *J. Phys. Chem.*, 117(15):695969, 2002.

[73] R. Heinrich and S. Schuster. *The Regulation of Cellular Systems*. Kluwer Academic Publishers, 1996.

[74] Iain Hepburn, Weiliang Chen, Stefan Wils, and Erik De Schutter. STEPS: efficient simulation of stochastic reactiondiffusion models in realistic morphologies. *BMC Systems Biology*, 6(36), 2012.

[75] A. V. Hill. The possible effects of the aggregation of the molecules of haemoglobin on its dissociation curves. *J. of Physiology*, 40, 1910.

[76] Wolfgang Hormann, Josef Leydold, and Gerhard Derflinger. *Automatic Nonuniform Random Variate Generation*. Springer-Verlag, 2004.

[77] M. Hucka and et al. The systems biology markup language (sbml): A medium for representation and exchange of biochemical network models. *Bioinformatics*, 19(4):524–531, 2003.

[78] M. Hucka and et al. Evolving a lingua franca and associated software infrastructure for computational systems biology: The systems biology markup language (sbml) project. *Systems Biology*, 1(41), 2004.

[79] D. A. Huffman. A method for the construction of minimum-redundancy codes. In *Proc. of IRE*, 1952.

[80] S. Indurkhya and J. Beal. Reaction factoring and bipartite update graphs accelerate the gillespie algorithm for large-scale biochemical systems. *PLoS One.*, 5(1):8125, 2010.

[81] A. Jansson and M. Jirstrand. The systems biology graphical notation. *Drug Discov. Today*, 15(9):36570, 2010.

[82] Matthias Jeschke and Adelinde M. Uhrmacher. Multi-resolution spatial simulation for molecular crowding. In *Proc. of Winter Simulation Conference*, 2008.

[83] Douglas W. Jones. An empirical comparison of priority-queue and event-set implementations. *Communications of the ACM*, 29(4):300–311, 1986.

[84] Thomas R. Kiehl, Robert M. Mattheyses, and Melvin K. Simmons. Hybrid simulation of cellular behavior. *Bioinformatics*, 20(3):316322, 2004.

[85] C. Kim, C. Cheng, S. Saldanha, and S. Taylor. Pka-i holoenzyme structure reveals a mechanism for camp-dependent activation. *Cell*, 130(6):1032–43, 2007.

[86] Haseong Kim and Erol Gelenbe. Stochastic gene expression modeling with Hill function for switch-like gene responses. *IEEE/ACM Trans. on Computational Biology and Bioinformatics*, 9(4):973–979, 2012.

[87] Hiroaki Kitano. *Foundation of Systems Biology*. MIT Press, 2001.

[88] Hiroaki Kitano. Computational systems biology. *Nature*, 420:206–210, 2002.

[89] Hiroaki Kitano. Systems biology: A brief overview. *Science*, 295:1662, 2002.

[90] Michael Klann, Arnab Ganguly, and Heinz Koepp. Hybrid spatial gillespie and particle tracking simulation. In *Proceeding of European Conference on Computational Biology (ECCB)*, 2012.

[91] D. Knuth. *The Art of Computer Programming*, volume 1. Addison-Wesley, 2011.

[92] H. Koeppl, D. Densmore, G. Setti, and M. di Bernardo. *Design and Analysis of Bio-molecular Circuits*. Springer-Verlag, 2011.

[93] W. Kolch. Meaningful relationships: the regulation of the ras/raf/mek/erk pathway by protein interactions. *Biochem. J.*, 351(2):289–305, 2000.

[94] Richard A. Kronmal and Arthur V. Peterson. On the Alias method for generating random variables from a discrete distribution. *The American Statistician*, 33(4):214–218, 1979.

[95] Yoshiki Kuramoto and Tomoji Yamada. Pattern formation in oscillatory chemical reactions. *Progress of Theoretical Physics*, 56(3):724–740, 1976.

[96] H. Kurata, H. El-Samad, T. M. Yi, M. Khammash, and J. Doyle. Feedback regulation of the heat shock response in e. coli. In *Proc. of CDC*, 2001.

[97] Thomas G. Kurtz. The relationship between stochastic and deterministic models for chemical reactions. *J. Chem. Phys.*, 57(2976), 1972.

[98] H. Kuwahara and I. Mura. An efficient and exact stochastic simulation method to analyze rare events in biochemical systems. *J. Chem. Phys.*, 29(16):165101, 2008.

[99] A. Lagnoux. Rare event simulation. *PEIS*, 20(1):45–66, 2006.

[100] S Lampoudi, D T. Gillespie, and L R. Petzold. The multinomial simulation algorithm for discrete stochastic simulation. *J. Chem. Phys.*, 130(9):94104, 2009.

[101] P. L'Ecuyer, V. Demers, and B. Tuffin. Rare events, splitting, and quasi-monte carlo. *ACM Transactions on Modeling and Computer Simulation*, 17(22), 2007.

[102] Trevor H. Levere. *Affinity and Matter: Elements of Chemical Philosophy 1800-1865*. Routledge, 1993.

[103] Hong Li and Linda Petzold. Logarithmic direct method for discrete stochastic simulation of chemically reacting systems. Technical Report, 2006.

[104] C. M. Lloyd, M. D. B. Halstead, and P. F. Nielsen. Cellml: its future, present and past. *Progress in Biophysics and Molecular Biology*, 85(2):433–450, 2004.

[105] L. Lok and R. Brent. Automatic generation of cellular reaction networks with moleculizer 1.0. *Nat. Biotec.*, 23(1):131–6, 2005.

[106] S. Mauch and M. Stalzer. Efficient formulations for exact stochastic simulation of chemical systems. *IEEE/ACM Trans. on Computational Biology and Bioinformatics*, 8(1):27–35, 2011.

[107] Carsten Maus, Stefan Rybacki, and Adelinde M Uhrmacher. Rule-based multi-level modeling of cell biological systems. *BMC Systems Biology*, 5(166), 2011.

[108] H. McAdams and A. Arkin. Stochastic mechanisms in gene expression. *Proc. Natl. Acad. Sci.*, 94:814819, 1997.

[109] H. McAdams and A. Arkin. It's a noisy business! genetic regulation at the nanomolar scale. *Trends in Genetics*, 15(2):65–69, 1999.

[110] James McCollum and *et al.* The sorting direct method for stochastic simulation of biochemical systems with varying reaction execution behavior. *Comp. Bio. Chem.*, 30(1):39–49, 2006.

[111] Donald A. McQuarrie. Stochastic approach to chemical kinetics. *Journal of Applied Probability*, 4(3):413–478, 1967.

[112] P. Metzner, C. Schtte, and E. Vanden-Eijnden. Illustration of transition path theory on a collection of simple examples. *J. Chem. Phys.*, 125(8):084110, 2006.

[113] Robin Milner. *Communicating and Mobile System: the π-Calculus*. Cambridge University Press, 1999.

[114] Ramon E. Moore, R. Baker Kearfott, and Michael J. Cloud. *Introduction to Interval Analysis*. SIAM, 2009.

[115] C. J. Morton-Firth and D. Bray. Predicting temporal fluctuations in an intracellular signalling pathway. *J. Theor. Biol.*, 192:117–128, 1998.

[116] B. Munsky and M. Khammash. The finite state projection algorithm for the solution of the chemical master equation. *J Chem Phys.*, 124(4):44104, 2006.

[117] I. Mura, D. Prandi, C. Priami, and A. Romanel. Exploiting non-markovian bio-processes. *Electronic Notes in Theoretical Computer Science*, 253(3):8398, 2009.

[118] N A. Neogi. Dynamic partitioning of large discrete event biological systems for hybrid simulation and analysis. In *In Proc. of Hybrid Systems: Computation and Control (HSCC)*, 2004.

[119] N. Le Novre and et al. The systems biology graphical notation. *Nat. Biotechnol.*, 27(8):73541, 2009.

[120] Nicolas Le Novre and Dominic Tolle. Particle-based stochastic simulation in systems biology. *Current Bioinformatics*, 1(3), 2006.

[121] Jrgen Pahle. Biochemical simulations: stochastic, approximate stochastic and hybrid approaches. *Brief Bioinform.*, 10(1):53–64, 2009.

[122] J. Pedraza and A. van Oudenaarden. Noise propagation in gene networks. *Science*, 307(5717):1965–1969, 2005.

[123] M F. Pettigrew and H. Resat. Multinomial tau-leaping method for stochastic kinetic simulations. *J. Phys. Chem.*, 126:084101, 2007.

[124] J. W. Pinney, D R.Westhead, and G. A. McConkey. Petri net representations in systems biology. *Biochem Soc Trans.*, 31(6):1513–5, 2003.

[125] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes: The Art of Scientific Computing*. SIAM, 3rd editon edition, 2006.

[126] J Puchalka and A M. Kierzek. Bridging the gap between stochastic and deterministic regimes in the kinetic simulations of the biochemical reaction networks. *J Biophys.*, 86(22):135772, 2004.

[127] A. Raj and A. van Oudenaarden. Single-molecule approaches to stochastic gene expression. *Annual Review of Biophysics*, 297:255270, 2009.

[128] Rajesh Ramaswamy, Nlido Gonzlez-Segredo, and Ivo F. Sbalzarini. A new class of highly efficient exact stochastic simulation algorithms for chemical reaction networks. *J. Chem. Phys.*, 130(24):244104, 2009.

[129] C V Rao and A P Arkin. Stochastic chemical kinetics and the quasi-steady-state assumption: Application to the gillespie algorithm. *J. Phys. Chem.*, 118(11):49995010, 2003.

[130] Christopher V. Rao, Denise M. Wolf, and Adam P. Arkin. Control, exploitation and tolerance of intracellular noise. *Nature*, 420:231–237, 2002.

[131] J. Raser and E. O'Shea. Noise in gene expression: Origins, consequences, and control. *Science*, 309(5743):2010–2013, 2005.

[132] M Rathinam, L R. Petzold, Y Cao, and D T. Gillespie. Stiffness in stochastic chemically reacting systems: The implicit tau-leaping method. *J. Phys. Chem.*, 119(24):1278494, 2003.

[133] W. Reisig. *Petri Nets: An Introduction, Monographs on Theoretical Computer Science*. Springer-Verlag, 1985.

[134] Haluk Resat, H. Steven Wiley, and David A. Dixon. Probability-weighted dynamic monte carlo method for reaction kinetics simulations. *J. Phys. Chem. B*, 105(44):1102611034, 2001.

[135] Robert Rnngren, Jens Riboe, and Rassul Ayani. Lazy queue: an efficient implementation of the pending-event set. In *Proc. of 24th Annual Symposium on Simulation (ANSS)*, 1991.

[136] H. Robbins and S. Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, 22(3):400, 1951.

[137] J. Vidal Rodrguez, Jaap A. Kaandorp, Maciej Dobrzynski, and Joke G. Blom. Spatial stochastic modelling of the phosphoenolpyruvate-dependent phosphotransferase (pts) pathway in escherichia coli. *Bioinformatics*, 22(15):1895–1901, 2006.

[138] Diego Rossinelli, Basil Bayati, and Petros Koumoutsakos. Accelerated stochastic and hybrid methods for spatial simulations of reactiondiffusion systems. *Chemical Physics Letters*, 451:1–3, 2008.

[139] G. Rubino and B. Tuffin. *Rare Event Simulation using Monte Carlo Methods*. Wiley, 2009.

[140] Howard Salis and Yiannis Kaznessis. Accurate hybrid stochastic simulation of a system of coupled chemical or biochemical reactions. *J. Phys. Chem.*, 122:054103, 2005.

[141] Werner Sandmann. Discrete-time stochastic modeling and simulation of biochemical networks. *Comput. Biol. Chem.*, 32(4):292, 2008.

[142] H. Sauro, A. Uhrmacher, D. Harel, M. Hucka, M. Kwiatkowska, P. Mendes, C. Shaffer, L. Stromback, and J. Tyson. Challenges for modeling and simulation methods in systems biology. In *In Proc. of Winter Simulation Conference*, pages 1720–1730, 2006.

[143] Harold A. Scheraga, Mey Khalili, and Adam Liwo. Protein-folding dynamics: Overview of molecular simulation techniques. *Annu. Rev. Phys. Chem.*, 58:57–83, 2007.

[144] Jeremy Schmutz and et al. Quality assessment of the human genome sequence. *Nature*, 429:365–368, 2004.

[145] Tim Schulze. Efficient kinetic monte carlo simulation. *J. Comp. Phys.*, 227(4):2455–2462, 2008.

[146] J A. Sekar and J R. Faeder. Rule-based modeling of signal transduction: a primer. *Methods Mol Biol.*, 880:139–218, 2012.

[147] Alexander Slepoy, Aidan P. Thompson, and Steven J. Plimpton. A constant-time kinetic monte carlo algorithm for simulation of large biochemical reaction networks. *J. Chem. Phys.*, 128(20):205101, 2008.

[148] James C. Spall. An overview of the simultaneous perturbation method for efficient optimization. *Johns Hopkins APL Technical Digest*, 19(4):482492, 1998.

[149] James C. Spall. *Introduction to Stochastic Search and Optimization: Estimation, Simulation, and Control*. John Wiley and Sons, Inc., 2003.

[150] Michael Stumpf, David J. Balding, and Mark Girolami. *Handbook of Statistical Systems Biology*. Wiley, 2011.

[151] Audrius B. Stundzia and Charles J. Lumsden. Stochastic simulation of coupled reactiondiffusion processes. *J. Comp. Phys.*, 127(168):196207, 1996.

[152] Zoltan Szallasi, Jrg Stelling, and Vipul Periwal. *Systems Modeling in Cell Biology: From Concepts to Nuts and Bolts*. MIT Press, 2010.

[153] K. Takahashi, S N. Arjunan, and M. Tomita. Space in systems biology of signaling pathways-towards intracellular molecular crowding in silico. *FEBS Lett.*, 579(8):1783–8, 2005.

[154] Kouichi Takahashi, Katsuyuki Yugi, Kenta Hashimoto, Yohei Yamada, Christopher J. Pickett, and Masaru Tomita. Computational challenges in cell simulation: a software engineering approach. *IEEE Trans. on Intelligent Systems*, 17(5):64–71, 2002.

[155] Gerald Teschl. *Ordinary Differential Equations and Dynamical Systems*. Graduate Studies in Mathematics, 2012.

[156] Vo H. Thanh and Roberto Zunino. Tree-based search for stochastic simulation algorithm. In *Proc. of ACM-SAC*, 2012.

[157] Vo H. Thanh and Roberto Zunino. Splitting for rare event simulation in biochemical systems. In *Proc. of SIMUTools*, 2013.

[158] M. Thattai and A. van Oudenaarden. Intrinsic noise in gene regulatory networks. *Proc. Natl. Acad. Sci.*, 98(15):86148619, 2001.

[159] T. Tian and K. Burrage. Binomial leap methods for simulating stochastic chemical kinetics. *J. Phys. Chem.*, 121(21):1035664, 2004.

[160] John J. Tyson, Katherine C. Chen, and Bela Novak. Sniffers, buzzers, toggles and blinkers: dynamics of regulatory and signaling pathways in the cell. *Current Opinion in Cell Biology*, 15(2):221–231, 2003.

[161] A.R. Tzafriri and E.R. Edelman. The total quasi-steady-state approximation is valid for reversible enzyme kinetic. *Journal of Theoretical Biology*, 226:303313, 2004.

[162] J. van Ommeren, M. Garvels, and D. Kroese. On the importance function in splitting simulation. *European Trans. on Telecommunications*, 13(4):363–371, 2002.

[163] Jeroen S. van Zon and Pieter Rein ten Wolde. Greens-function reaction dynamics: A particle-based approach for simulating biochemical networks in time and space. *J. Chem. Phys.*, 123:234910, 2005.

[164] E. Vanden-Eijnden and M. Venturoli. Revisiting the finite temperature string method for the calculation of reaction tubes and free energies. *J. Chem. Phys.*, 130:194103, 2009.

[165] E. Vanden-Eijnden, M. Venturoli, G. Ciccotti, and R. Elber. On the assumptions underlying milestoning. *J. Chem. Phys.*, 129:174102, 2008.

[166] J. Vilar, H. Kueh, N. Barkai, and S. Leibler. Mechanisms of noise-resistance in genetic oscillators. *Proc. Natl. Acad. Sci.*, 99(9):156–161, 2002.

[167] Michael D. Vose. A linear algorithm for generating random numbers with a given distribution. *IEEE Trans. on Software Engineering*, 17(9):972–974, 1991.

[168] P. Waage and C. M. Gulberg. Studies concerning affinity. *Christiana*, 1864.

[169] Alastair J. Walker. An efficient method for generating discrete random variables with general distributions. *ACM Trans. on Mathematical Software (TOMS)*, 3(3):253–256, 1977.

[170] P. Warren and P. ten Wolde. Enhancement of the stability of genetic switches by overlapping upstream regulatory domains. *Phys. Rev. Lett.*, 92(12):128101, 2004.

[171] Darren J. Wilkinson. *Stochastic Modeling of System Biology*. Chapman & Hall/CRC, 2006.

[172] Verena Wolf, Rushil Goel, Maria Mateescu, and Thomas A Henzinger. Solving the chemical master equation using sliding windows. *BMC Systems Biology*, 4(42), 2010.

[173] Olaf Wolkenhauer and Mihajlo Mesarovic. Feedback dynamics and cell function: Why systems biology is called systems biology. *Mol Biosys.*, 1(1):14–16, 2005.